NICOR MCU

MCU USER GUIDE

Master Control Unit

USER GUIDE



NICOR

NICORLIGHTING.COM/**SOLUTIONS/IMS**

Contents

1 MCU Overview 9
1.1 Product overview 9
1.2 The MCU and the IMS controller 9
1.3 Scale and multi-tenancy 10

2 System Architecture 11
2.1 The four applications 11
2.2 How configuration and status flow 11
2.3 The Autonomy Principle 12
2.4 Resilience: when the Gateway is down 12
2.5 Resilience: when the DataManager is down 12

3 The System Hierarchy 13
3.1 Clients and Locations 13
3.2 Buildings 14
3.3 Areas — the self-contained control unit 14
3.4 Controllers, Networks and Zones 14
3.5 Fixtures and Input Devices 14
1 The Zone at the Center of Everything 16
2 What Is a Zone? 16
3 Zone Settings and Inheritance 17
4 The Lighting Model: Four Zone States 18
5 State Lighting Strategies 18
6 Transitions and Timeouts 19
7 Zone Inputs: Turning Sensing into Control 20
8 How the Zone Controls Its Fixtures 22
9 How the Zone Controls Relays 23
10 Seeing the Zone in Action 24
11 The Flexible Heart 25

5 Cross-Cutting Conventions 26
5.1 The warning and status model 26
5.2 The inheritance convention 26
5.3 A DALI-2 primer 26
5.4 Licensing overview 27

6 Getting Started 28
6.1 Logging in and choosing a server 28
6.2 Remembering your login and renewing certificates 29
6.3 Selecting a building 30

7 The Building View 31
7.1 Orientation 31
7.2 Reading the status bars 31
7.3 Warnings 32

7.4 Controller status 32
7.5 The File and Gateway menus 33

8 Working in an Area 34
8.1 Opening an area 34
8.2 Reading the floorplan 35
8.3 Live values 36
8.4 Hovering for detail 36

9 Manual Control 38
9.1 Setting a zone's state 38
9.2 Broadcasting a level across a network 38
9.3 Triggering an input event 39

10 Energy Reporting 40
10.1 Opening the report 40
10.2 Two ways to look at consumption 40
10.3 Choosing what to measure 41
10.4 Gaps in the data 42

11 What Your Permissions Allow 43
11.1 Two questions: what, and where 43
11.2 The five permissions 43
11.3 Why something is greyed out 43

12 Commissioning a New System 45
12.1 Which path are you on? 45
12.2 Preparation (both paths) 45
12.3 Setting up the building 46
12.4 Path A — adopting an existing IMS 47
12.5 Path B — a fresh installation 47
12.6 Combining the two 47
12.7 Keeping it in step afterwards 47

13 Building Configuration 49
13.1 Identity and location 49
13.2 The gateway 49
13.3 Email notifications 50
13.4 Zone types 51
13.5 Building fixture defaults 51
13.6 Demand response 52
13.7 ADR configuration 52
13.8 Floorplan drawings 53

14 The Area Editor in Depth 54
14.1 The editor window 54
14.2 Creating and selecting objects 56
14.3 Linking objects to zones 56
14.4 Copying and pasting 58
14.5 Moving objects 58
14.6 Deleting objects 59

14.7 Area properties and sheets 60
14.8 Printing a sheet 61
14.9 Exporting an area 62

15 Controllers and Networks 64
15.1 Where a controller sits 64
15.2 The controller's properties 64
15.3 The networks on a controller 65
15.4 Inside a network 66
15.5 Network mode and the two-tier update policy 68
15.6 Lighting controls 69
15.7 Synchronization status 70
15.8 Administration and engineering tools 70
15.9 The network update menu 71
15.10 Bringing a controller up 72

16 Configuring Zones 73
16.1 General properties 73
16.2 Zone linking 73
16.3 Where settings come from: zone types 74
16.4 Zone settings — the state matrix 75
16.5 Demand response and fixture defaults 76
16.6 Zone Inputs 76
16.7 Fixtures and relays 77
16.8 Scenes 78
16.9 History 79

17 Fixtures and Input Devices 80
17.1 Owned, and shared 80
17.2 A fixture's properties 80
17.3 Fixture settings, and when to use them 82
17.4 Fixture scenes 83
17.5 Fixture addressing 84
17.6 An input device's details 84
17.7 Instance settings 86
17.8 Properties and addressing 86
17.9 Working on the canvas 87
17.10 Triggering an input by hand 89

18 Device Reconciliation and Addressing 90
18.1 What reconciliation is for 90
18.2 Reading the addressing grid 90
18.3 Duplicate addresses 92
18.4 When the counts do not match 93
18.5 The four mechanisms 94
18.6 Mechanism 1 — Identify & Assign 95
18.7 Mechanism 2 — Update addresses from matching product IDs 96
18.8 Mechanism 3 — Update network or controller from IMS 96
18.9 Mechanism 4 — Scan network for new devices 98

18.10 Re-addressing the bus 100
18.11 Working on a single device 101
18.12 Choosing a mechanism 101

19 IMS ↔ MCU Synchronization 103
19.1 Two models of one installation 103
19.2 The revision model 103
19.3 Controller status at a glance 104
19.4 The two-tier sync policy 104
19.5 Forcing a resync 105
19.6 Reconciling the device model 105
19.7 From commissioning to steady state 106

20 Scheduling 107
20.1 How scheduling works here 107
20.2 The schedule calendar 107
20.3 The events list 107
20.4 Defining when an event runs 108
20.5 What an event does 109
20.6 Which zones an event affects 109
20.7 Priority and overlapping events 110
20.8 Schedules and the controllers 110

21 Demand Response and Daylight Harvesting 111
21.1 Two ways to use less light 111
21.2 What demand response is for 111
21.3 Setting what a zone gives up 111
21.4 Driving demand response 113
21.5 What daylight harvesting is 114
21.6 Tuning the loop 114
21.7 How the layers combine 115
21.8 External demand response (OpenADR) 115

22 The NICOR Administration App 117
22.1 What the app is for 117
22.2 How it is organized 117
22.3 The System branch 117
22.4 The Clients branch 118
22.5 Who can use it 118

23 Users and Permissions 119
23.1 The user list 119
23.2 Creating and editing an account 119
23.3 Granting permissions 120
23.4 Placing the account in the hierarchy 121
23.5 Escalation is prevented 121

24 Clients, Locations and Buildings 122
24.1 The client record 122
24.2 Locations 122

24.3 Buildings 123
24.4 Importing and exporting buildings 124
24.5 Setting up a new customer 124

25 Licensing 125
25.1 How licensing is structured 125
25.2 What the permissions enable 125
25.3 Applying a license 126
25.4 Status, expiry and when it takes effect 126

26 Software and Gateway Registration 128
26.1 Publishing software 128
26.2 Registering a gateway service 129
26.3 How registration fits together 129

27 The Device Catalog 131
27.1 Why the catalog matters 131
27.2 Manufacturers 131
27.3 Fixture driver types 132
27.4 Fixture load types and power 133
27.5 Input device types 134
27.6 Import and export 135

28 The Data Manager 136
28.1 What it holds 136
28.2 Settings 136
28.3 The installers page 137
28.4 Creating the database 138
28.5 Backup and restore 138
28.6 Long-term storage and audit 139

29 The Gateway 140
29.1 What the gateway does 140
29.2 Network and identity 140
29.3 Operating options 141
29.4 Email 141
29.5 Notifications 141
29.6 Assigned buildings 142

30 Security and Certificates 143
30.1 How connections are secured 143
30.2 Trust on first use 143
30.3 When a certificate changes 143
30.4 Renewing trust 144
30.5 Practical guidance 144

31 Data Retention 145
31.1 The whole picture 145
31.2 Configuration is permanent 145
31.3 Energy data moves rather than expires 145
31.4 Zone state history is transient 146

31.5 The audit log 146
31.6 What this means for backups 146

32 Monitoring and Diagnostics 147
32.1 The gateway report 147
32.2 Reading it 147
32.3 What healthy looks like 148
32.4 Log files 148
32.5 The audit log 148
32.6 A diagnostic order 148

Appendix A 150
A.1 What is exposed 150
A.2 Enabling and connecting 150
A.3 The object list 151
A.4 Zone state — the present value 152
A.5 Status flags 152
A.6 Writing to a zone 153
A.7 Change of value 154
A.8 Before commissioning an integration 154

Appendix B 155
B.1 Shape and color 155
B.2 Zone markers 155
B.3 Zone-state symbols 156
B.4 Controller icons 157
B.5 Fixture icons 158
B.6 Input devices and their instances 159
B.7 Reading color 160

Appendix C 161
C.1 What DALI is, and what it is not 161
C.2 The bus: wiring, power and speed 161
C.3 Addressing: short addresses, groups and broadcast 162
C.4 Control gear and control devices 162
C.5 Levels, the dimming curve and fades 163
C.6 Scenes 165
C.7 Instances, events and the reason timeouts exist 165
C.8 Identity: GTIN and Product ID 166
C.9 How addresses get assigned 166
C.10 How MCU concepts map onto DALI 167
C.11 The IEC 62386 parts 168
C.12 Glossary 168

Appendix D 171
D.1 The parts, in one model 171
D.2 Reaching the VTN 171
D.3 Authenticating to the VTN 171
D.4 Identifying this VEN 172
D.5 How the VEN is notified 172

D.6 Report confirmation 173
D.7 What the building does with a level 173

Appendix E 175

E.1 Before anything else 175
E.2 Nothing appears to be working 175
E.3 A change will not take effect 175
E.4 The wrong lights respond 176
E.5 Occupancy and timing 176
E.6 Daylight harvesting and demand response 177
E.7 Devices and addressing 178
E.8 Energy, history and backups 178
E.9 Questions that come up repeatedly 178

PART I · INTRODUCTION & CORE CONCEPTS

1 MCU Overview

This chapter introduces the NICOR MCU — what it is, the controller platform it manages, and the scale at which it operates. It lays the groundwork for the architecture and hierarchy chapters that follow.

1.1 Product overview

The NICOR MCU (Master Control Unit) is a software suite for managing DALI-2 lighting installations at scale. It provides a single place to configure, monitor and coordinate lighting across many buildings and sites, while the real-time control of each installation is carried out locally by NICOR IMS controllers. In short, the MCU is the **management and coordination layer** that sits above a fleet of autonomous lighting controllers.

The suite is delivered as four cooperating applications — a central data service (the **DataManager**), one or more site **Gateways**, an end-user operations app (**Site Manager**), and a platform administration app (the **NICOR Administration App**). Chapter 2 describes how these fit together; for now the important idea is that the MCU is not a single program but a small ecosystem, with a clean separation between the data it holds centrally, the gateways that deliver that data to site, and the controllers that execute it.

Because a single MCU deployment is designed to serve many independent customers, it is **multi-tenant** from the ground up: clients, their locations and their buildings are kept separate, and access is governed by role-based permissions. This makes it equally suited to a single organization managing its own properties and to a service provider managing lighting on behalf of many customers.

1.2 The MCU and the IMS controller

Understanding the MCU begins with understanding its relationship to the **IMS controller**, because the two divide the work of a lighting system between them.

The IMS controller is the hardware that connects directly to the DALI-2 bus. It addresses the physical devices, runs the occupancy and scheduling logic, and drives the fixtures in real time. Crucially, it does this **autonomously**: once configured, an IMS continues to run its schedules and respond to its sensors entirely on its own, with no dependence on the MCU being reachable. If the network above it goes away, the lights keep working.

The MCU, by contrast, does not drive lights moment to moment. Its role is to hold the authoritative configuration, to coordinate many controllers and buildings from one place, and to provide monitoring and reporting. A useful way to frame the split is that the MCU **models intent** — which zones exist, how they should behave, what the schedules are — while the IMS **executes** that intent on the bus. The MCU authors and stores the design; the gateway carries it to the controller; the controller runs it.

> **The core division of labor**
>
> The MCU *models* a lighting installation; the IMS controller *runs* it. Everything else in this manual — configuration, synchronization, monitoring — follows from that split. It is also why the system is resilient: the one thing that must never fail, real-time lighting control, lives on the most local, self-sufficient component, while the convenience of central management sits above it and can tolerate interruption without the lights being affected.

1.3 Scale and multi-tenancy

The MCU organizes everything it manages into a single strict hierarchy, from the customer at the top down to the individual lighting fixtures and control devices. A **client** owns one or more **locations**; a location contains **buildings**; a building is divided into **areas**; each area contains **controllers**; each controller hosts DALI-2 **networks**; each network is divided into **zones**; and each zone groups the **fixtures and input devices** it controls. Chapter 3 walks through each level in turn; the point here is that this one hierarchy scales from a single small building to a large distributed set of properties without changing shape.

Some limits are inherited from DALI-2 itself and apply at the network level — sixteen zones, sixty-four fixtures and sixty-four input devices per network. These are building blocks rather than ceilings on the system: many networks make up a controller, many controllers a building, and many buildings a client, so an installation as a whole scales far beyond any single network's limits.

Multi-tenancy runs through the whole hierarchy. Different clients' data is kept separate, and what any given person can see or do is governed by a small set of **permission flags** (covered in Part IV). A standard user might operate a single building through the Site Manager app; an engineer might commission and configure controllers; a platform administrator might manage clients, licensing and the backend services. The same hierarchy and the same MCU serve all three — each person sees the part appropriate to their role, which is exactly why this manual is organized by audience.

2 System Architecture

This chapter shows how the four applications introduced in Chapter 1 fit together, how configuration and status move between them, and — importantly — how the system behaves when parts of it are unavailable. The recurring theme is the autonomy principle: control lives low, management lives high.

2.1 The four applications

The MCU is made up of two backend **services** and two user-facing **applications**:

Component	Type	Role
DataManager	Service	The authoritative central data store. Holds the configuration database (clients, buildings, zones, users, licenses, schedules), the long-term storage archive for energy data, and the audit log. Everything else ultimately reads from and writes to it.
Gateway	Service	The on-site mediator between the MCU and the IMS controllers. It loads a building from the DataManager, pushes configuration down to the controllers, reads live status back, and handles notifications, email and BACnet exposure. A deployment can run several gateways.
Site Manager	Application	The day-to-day front-end. Users log in, pick a building, monitor and operate it, and view energy reports; engineers use the same app to build floorplans and configure controllers, networks and zones.
NICOR Administration App	Application	Platform administration: managing users and permissions, clients, locations and buildings, licensing, software packages, gateway registration, and the global device catalog.

Each of the two services also has its own small **administration utility** — the DataManager Administration App and the Gateway Administration App — used to install, configure and maintain the service itself (ports, certificates, the database, email and so on). These are covered in Part IV.

2.2 How configuration and status flow

The system has a clear structure: **configuration flows down, status flows up**. The path runs through each component in turn:

DataManager → Gateway → IMS controller → DALI-2 devices

Configuration is authored in Site Manager (or, for platform-level data, the NICOR Administration App) and saved to the DataManager. A gateway loads its building from the DataManager and delivers the configuration down to the IMS controllers, which in turn drive the DALI-2 devices on the bus. Status and telemetry return along the same chain in reverse: the devices report to the IMS, the gateway reads that status from the controller, live values surface in Site Manager, and historical data is stored back in the DataManager.

This flow is not continuous chatter. *When* an update travels down — and which way it should travel when the two sides differ — is governed by a revision-numbering model, described in full in Part III. For this chapter the shape of the flow is what matters: a single downward path for intent and a single upward path for status.

2.3 The Autonomy Principle

The single most important architectural idea in the MCU is that the **IMS controller runs independently**. Once a controller has been configured and has loaded its building, it executes its schedules and occupancy logic entirely on its own hardware. It does not consult the gateway or the DataManager to decide what the lights should do next — that decision-making lives on the controller.

The consequence is that the MCU is deliberately *not* in the real-time control loop. The gateway and DataManager provide configuration, coordination and visibility, but the lights do not depend on them second to second. This is what makes the layers above the controller safe to interrupt — the subject of the next two sections.

2.4 Resilience: when the Gateway is down

If a gateway becomes unreachable, the IMS controllers beneath it **carry on unaffected** — they already hold their configuration and keep running autonomously. What is lost is the live link: Site Manager can still load and edit a building's configuration (that comes from the DataManager, not the gateway), but **live status and control are unavailable**, shown by a red "Live status unavailable" banner across the affected objects.

Edits made during a gateway outage are saved normally, but they will not reach the controllers until the gateway is restored and the usual synchronization resumes. In other words, you can keep working on the design; it simply queues until the delivery path is restored.

2.5 Resilience: when the DataManager is down

The DataManager is the source of all configurations, if it is unavailable, the front-end applications — Site Manager and the administration apps — **cannot log in or load buildings**, because there is nothing to load.

However, Gateways that were *already running* keep going: they have their buildings loaded and continue to run independently, and the IMS controllers below them are of course unaffected. Two things are lost during the outage — historical energy data cannot be stored, and a gateway that needs to *reload* (after a restart or a dropped connection) will fail to do so until the DataManager service is restored.

The overall picture is symmetrical and easy to hold in mind:

Component down	IMS controllers	Running Gateway	Site Manager / Admin apps
Gateway	Run normally (autonomous)	—	Config viewable and editable; no live status or control
DataManager	Run normally (autonomous)	Keep running on loaded config; no history stored; cannot reload	Cannot log in or load buildings

> **The pattern: autonomy increases with depth**
>
> Reading the architecture from the top down, each layer depends less on the one above it. The DALI devices depend on their IMS; the IMS depends on nothing above it to keep running; the gateway can run without the DataManager once loaded; and only the front-end applications truly need everything present. The closer a component is to the light itself, the more self-sufficient it is — which is exactly the property you want in a lighting-control system.

3 The System Hierarchy

The MCU arranges everything it manages into one hierarchy, from the customer at the top down to the individual light fitting. This chapter walks each level in turn — what it represents and the rules that bind it to its neighbors.

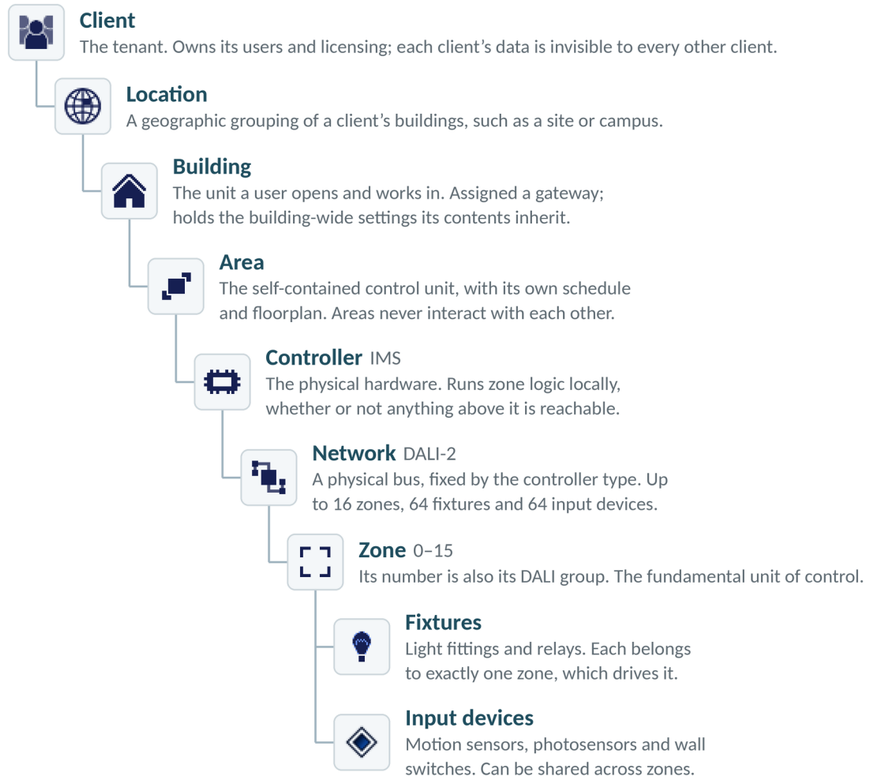


Figure 3.1. *The system hierarchy, from the client at the top to the individual DALI-2 device.*

3.1 Clients and Locations

At the top sits the **client** — the tenant. Everything in the system ultimately belongs to a client, and multi-tenant separation is enforced at this level: one client's data is invisible to another. A client owns its **licenses** (which are always issued per-client) and its **users**.

Beneath the client are **locations** — geographic groupings that organize a client's buildings, for example by site or campus. Locations are an organizational convenience; the real configuration begins one level down, at the building.

3.2 Buildings

A **building** is the principal unit of a lighting installation — the thing a user opens and works in. Each building is assigned a **gateway**, which delivers its configuration to the controllers, and it carries the building-wide settings that its contents inherit from: the **zone types** available to its zones, default fixture settings, demand-response definitions, and notification and email setup.

A building also holds its geographic **coordinates** (latitude and longitude), which the controllers use to compute local sunrise and sunset times for scheduling. A building is divided into areas.

3.3 Areas — the self-contained control unit

The **area** is where control becomes self-contained, and it is one of the most important levels to understand. Each area is an **independent unit**: the objects inside one area do not interact with those in another, each area has its own **schedule**, and each has its own **floorplan** (which may span several sheets).

This independence is deliberate. It lets a large building be divided into parts that are configured, scheduled and evaluated separately — and, as Part III shows, it is also the unit at which changes are tracked and synchronized. An area contains one or more controllers.

3.4 Controllers, Networks and Zones

Within an area, each **controller** is an IMS controller — the physical hardware described in Chapter 1. A controller hosts a fixed number of DALI-2 **networks** determined by its type; these are physical children, created with the controller and not added or removed individually.

Each network is divided into **zones**, numbered 0 to 15. As Chapter 4 explains, a zone's number is also the DALI group its fixtures belong to — which is precisely why a network supports exactly sixteen zones. The zone is the fundamental unit of control in the whole system, and Chapter 4 is devoted entirely to it.

3.5 Fixtures and Input Devices

At the bottom of the hierarchy are the physical DALI-2 devices, of two kinds. **Fixtures** are the light fittings (and relays); each fixture belongs to exactly *one* zone and is driven by it. **Input devices** — motion sensors, photosensors and wall switches — provide the triggers and readings that drive zone behavior, and unlike fixtures a single input device can be **shared across several zones**.

Both kinds are identified by a **GTIN** that maps them to an entry in the global device catalogue, and each carries a DALI **short address** (0–63) that binds the MCU's model to the real device on the bus — the addressing that Part III's reconciliation mechanisms exist to establish.

> **Owned versus shared — a distinction to remember**
>
> A fixture is *owned* by exactly one zone; an input device is *shared*. This single difference echoes throughout the manual — it governs how objects are copied, how they move when you drag a zone, and what is removed when you delete one. Wherever the two behave differently, ownership is the reason.

s

NICOR MCU

MASTER CONTROL UNIT

Understanding Zones

The Flexible Heart of the Control System

Concept & Configuration Guide

1 The Zone at the Center of Everything

In the NICOR MCU, the zone is where lighting decisions are made. Everything above it in the system — clients, buildings, areas, controllers and networks — exists to organize the installation and deliver commands to the right place. Everything below it — the individual fixtures and sensors — exists to sense the environment and physically act on it. The zone is the layer in between, where sensing becomes decision and decision becomes light.

A zone is more than a group of lamps switched together. It is a small, self-contained control machine: it holds a model of whether its space is occupied, it decides how brightly to light that space and by what method, it listens to sensors and switches, and it drives its fixtures — and any relays — accordingly. Because each zone runs this logic autonomously on its IMS controller, the behavior keeps working even if the gateway or the wider network is disconnected.

This guide explains the whole model: how zones are structured and why that structure is beneficial, the four-state lighting model and the reasoning behind it, the three lighting strategies a state can use, how transitions and timeouts create smooth automation, how the zone input matrix turns sensor activity into control, and finally how the zone commands its fixtures and relays. By the end it should be clear why the zone is described as the flexible heart of the system.

2 What Is a Zone?

A **zone** is a logical grouping of fixtures on a single DALI-2 network that are controlled together as one entity. Rather than addressing lamps one at a time, the system treats the zone as the unit of control — you configure behavior once, at the zone, and every member fixture follows.

2.1 Zone number = DALI group

Each zone carries a **zone number from 0 to 15**. That number is not just a label: it is the DALI group that every member fixture is bound to. When the zone needs to change light level, the controller issues a single DALI group-broadcast command to that group, and all the fixtures respond at once. This is efficient and fast, but it also imposes a natural ceiling — DALI provides exactly 16 groups, so a network supports exactly **16 zones**.

> **The one exception**
>
> Fixtures whose **Assigned role is Relay** are deliberately kept out of the zone's DALI group and driven by separate, direct commands instead. This lets their on/off state be controlled independently of the lighting level — covered in full in section 9.

2.2 Where the zone sits in the hierarchy

The control hierarchy flows downward from the building to the individual device:

Building → Area → Controller (IMS) → DALI-2 Network → **Zone** → Fixtures / Input Devices

An area is a self-contained control unit with its own schedule; the objects inside one area do not interact with those in another. Within an area, each controller manages up to two DALI-2 networks, and each network hosts its zones. A fixture belongs to exactly one zone, but — as we will see — a single input can drive several zones at once.

2.3 Network limits

The system enforces the following hard limits per DALI-2 network. Drag-and-drop and copy/paste are both blocked once a limit is reached; there is no override, because these are physical constraints of the DALI bus.

Resource	Limit	Why
Zones per network	**16**	One per DALI group
Fixtures per network	**64**	DALI short-address range 0–63
Input devices per network	**64**	DALI short-address range 0–63
Scenes per zone	**16**	DALI scene range 0–15

2.4 Enabling, naming and the floorplan marker

A zone can be individually **enabled or disabled** — when disabled, the controller issues no lighting commands for it and its member fixtures simply hold their last level. A **Preserve state** option lets a zone restore its last state after an IMS power cycle rather than starting fresh. Each zone also carries a free-text comment field for engineering notes.

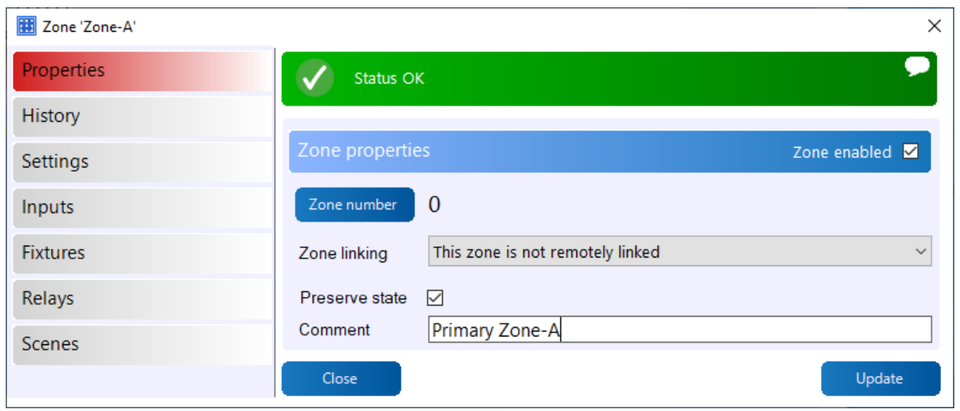
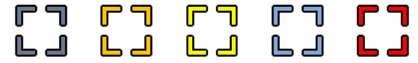


Figure 1. *Zone Properties — general settings, including the zone number picker (slots 0–15), the enable toggle and Preserve state.*

On the area floorplan, a zone is drawn as a bracket marker whose **color reflects its current state**. If the parent IMS controller is not connected to the MCU and the state is unknown, the marker is shown in red:



Unoccupied Preliminary Occupied Timeout Unknown

3 Zone Settings and Inheritance

Every zone inherits from a **Zone Type** — a reusable template defined at building level. A zone must always name a parent zone type (the system guarantees at least one exists in every building), and it takes any unset value from that type. This means common behavior can be defined once on a zone type such as "Office" or "Corridor" and shared by many zones, while any individual zone can still override specific values where it needs to differ.

The inheritance convention is used consistently throughout the application: a **green left-arrow** means the value is set explicitly at this level, while a **red up-arrow with the value in red** means it is inherited from the parent. Clicking the arrow toggles between the two. An **Inherit all** button resets everything back to inherited in one action.

A zone's settings are organized into three groups — the state matrix (Zone settings), Demand Response levels, and default Fixture settings for its members — all reachable from the zone's Settings tab and all following the same inheritance rules.

4 The Lighting Model: Four Zone States

At any moment a zone is in exactly one of four states. The state determines how the zone lights its space. The four states are not arbitrary — they map onto the natural narrative of a space being approached, used, and vacated.

	State	Meaning
	Unoccupied	The resting floor state — nobody present, minimum energy (often off or a low background level). Nothing times out from here; it is the bottom of the decay chain.
	Preliminary	A transitional, pre-arrival state. Someone is nearby or approaching, so the space is brought to a safe, welcoming level before they commit to entering it.
	Occupied	Presence is confirmed — the space is actively in use and receives full working light by whichever method the state is configured to use.
	Timeout	A grace / decay state entered when activity appears to have stopped. It holds a lower level for a while before finally falling back to Unoccupied.

4.1 Why four states, and not simply on / off?

Real occupancy lighting has to feel smooth and safe to a person while still saving energy. Two of the four states exist specifically to guard the moments of transition into and out of a space.

The *exit* transition is guarded by **Timeout**, and its existence comes directly from a hardware reality: a DALI-2 motion sensor only ever reports "motion detected" — there is no "motion stopped" event. The system therefore cannot know the instant a room empties; it can only notice the *absence* of fresh motion over time. So instead of cutting the lights the moment motion stops being reported, the Occupied state runs a timer, and when that timer expires the zone drops to Timeout — a lower "are you still there?" level — rather than straight to darkness. If the space is in fact still occupied, any small movement re-triggers Occupied; if not, Timeout's own short timer expires and the zone finally settles to Unoccupied. This two-stage decay is what stops the lights going out on someone sitting still.

The *entry* transition is guarded by **Preliminary**, which lets a space be lit to a safe level before an occupant has actually arrived, without the energy cost — or, in shared areas, the disruption — of jumping straight to full Occupied lighting somewhere nobody has gone yet.

4.2 Preliminary in practice: the corridor example

Imagine a main corridor that splits into a left branch and a right branch. A sensor in the main corridor can be used to bring both branches up to a gentle Preliminary level as soon as someone enters the main corridor — so that whichever way the person turns, they never step into an unlit space. When they actually turn (say) left, the occupancy sensor in the left branch picks them up and escalates that branch to full Occupied light. The right branch, never having received its own confirming trigger, simply times out gracefully back to Unoccupied.

> **Why this matters**
>
> The occupant is never led into darkness, no branch is fully lit speculatively, and the unused branch fades out on its own. This is lighting automation working *with* the human rather than lagging behind them — a direct benefit to both safety and occupant wellbeing.

5 State Lighting Strategies

Each state (other than Unoccupied, which has no transition of its own) independently chooses one of three lighting methods. Because the choice is per-state, a single zone can, for example, daylight-harvest while occupied but drop to a plain

fixed level while timing out.

Fixed Power

The simplest method: the zone drives its fixtures to a set arc-power level, expressed both as a raw DALI value and a percentage. It needs no sensors and is completely predictable — ideal for spaces where a known, constant level is wanted.

Light Levels (daylight harvesting)

A closed-loop method: the zone targets a light level in ftc / lux, measured by one or more photosensors linked to the zone (multiple readings are averaged). The controller continuously "hunts" — nudging arc power up or down — to hold the space at the target as daylight rises and falls, trimming energy use when natural light is plentiful. Two controller-wide tuning parameters govern the hunt: an adjustment interval (how often it re-checks) and a target tolerance (how close to the setpoint counts as close enough to stop adjusting).

Scenes

The zone recalls a pre-defined scene — a named-by-number arrangement in which each member fixture has its own stored level. Scenes are ideal where different looks are wanted from the same fixtures (for example a meeting room's presentation, discussion and cleaning scenes). Scenes are covered in detail in section 8.3.

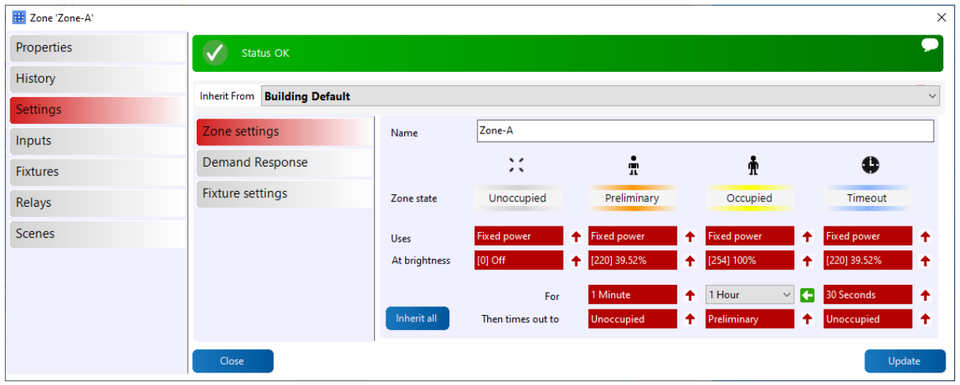


Figure 2. *The Zone settings matrix: one column per state, each with its lighting method (Uses), level or target (At brightness), timeout duration (For) and destination state (Then times out to). Red values are inherited from the zone type; green are set explicitly.*

6 Transitions and Timeouts

Each state (except Unoccupied) specifies two more things alongside its lighting: a **timeout duration** ("For…") and a **destination state** ("Then times out to…"). When a state's timer expires, the zone moves to the destination state. Crucially, this is fully configurable — any state can be set to time out to any other state — so you are not locked into a fixed sequence. The only fixed rule is that nothing times out *from* Unoccupied, because it is the resting floor.

On creation, every state defaults to timing out to Unoccupied. A typical configuration then wires the natural decay chain **Occupied →Timeout →Unoccupied**, giving the two-stage graceful fade described in section 4.1. Because the durations and destinations are yours to set, the same machine can model a fast-cycling corridor or a slow, forgiving conference room.

Reset timeout

A special input action, **Reset timeout**, restarts whichever state's timer is currently running — the "I'm still here" action. Combined with the input matrix (next section), it is how fresh motion in an occupied room keeps the lights on indefinitely without ever re-issuing a level change.

7 Zone Inputs: Turning Sensing into Control

States would be inert without something to move the zone between them. That something is the zone's inputs. Inputs come from DALI-2 input devices, and a single physical device can expose several independent instances. There are three input types:

- **Motion sensor** — a single "Motion Detected" event (there is no vacancy event; departure is handled by timeout decay). This is the primary automatic driver of occupancy.
- **Photosensor** — passive. It does not trigger anything; it contributes a light-level reading that feeds daylight harvesting, the canvas live values and the history graph. It is an input to the algorithm, not a trigger.
- **Button** — three events (Short press, Long press, Double press), providing manual override and user control.

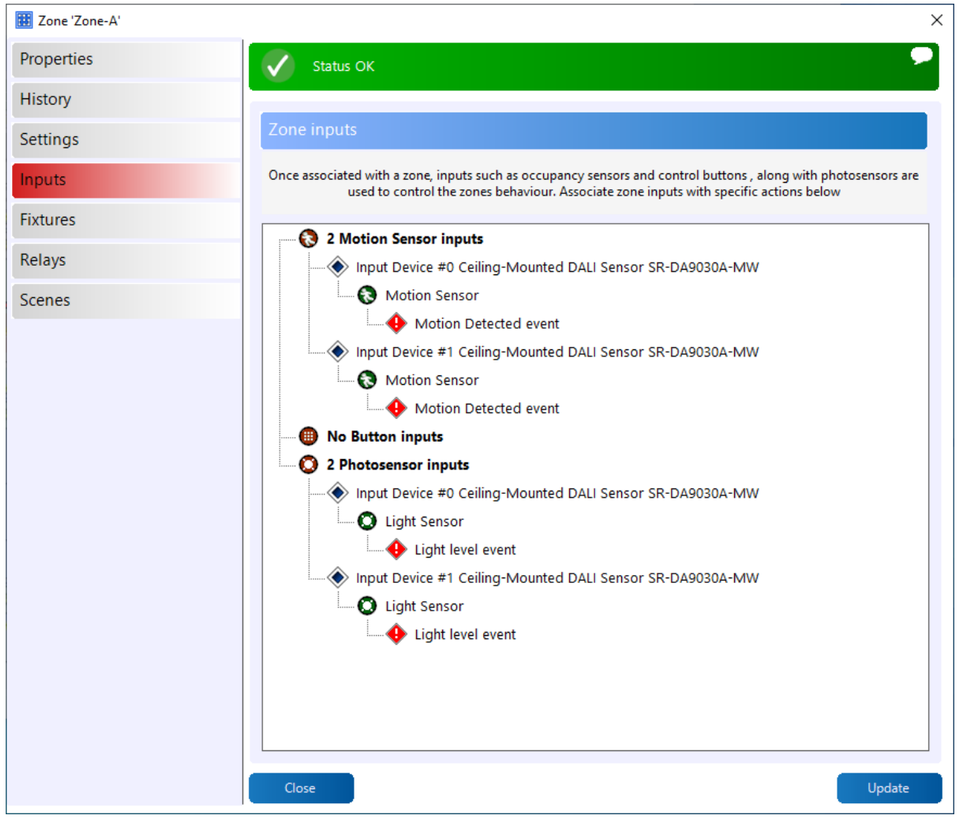


Figure 3. *The Inputs tree for a zone, grouping each contributing instance under its type (Motion Sensor, Button, Photosensor) with its configured event(s) beneath.*

7.1 The action matrix

The mechanism that ties an input to the state machine is the action matrix. Every switching event — the motion event and each button event — carries a matrix with one column per current zone state. In other words, the response to an input depends on the state the zone is already in.

Each cell of the matrix chooses from the following actions:

No Action	Set Unoccupied	Set Preliminary
Set Occupied	Set Timeout	Select next scene
Dim zone	Brighten zone	**Reset timeout**

This state-dependence is what makes the system expressive. A motion event might be set to Set Occupied when the zone is Unoccupied, Preliminary or Timeout, but Reset timeout when the zone is already Occupied — so fresh movement wakes a dark room yet merely re-arms the timer in a room that is already lit. A single physical button, through three different events each with their own matrix, can wake a zone, cycle its scenes and re-arm its timer depending on context.

7.2 Composing behavior: back to the corridor

The corridor example reveals exactly why the matrix is a grid rather than a single "input → action" mapping. It would be wrong to wire the main-corridor sensor to bluntly "Set Preliminary" on a branch, because if that branch were already Occupied — someone working in it, fully lit — a blunt command would drag it down to the lower level.

Instead you configure the branch zone's response to that trigger differently in each state, so the net effect is "ensure at least Preliminary, but only where the branch is currently unlit, and never disturb a higher state that is already active":

	If branch is…	Do this	Reasoning
	Unoccupied	**Set Preliminary**	Bring the dark branch up to a safe path level — the actual goal.
	Preliminary	No Action	Already there; nothing to do.
	Occupied	No Action	Leave the fully-lit, actively-used branch alone — do not dim it.
	Timeout	No Action	Let its own decay continue undisturbed.

The matrix has no literal "ensure minimum" verb — you *compose* that behavior by choosing the per-state action deliberately, using No Action in the states you want to protect. A single-action model could never express "raise it if it's off, but don't touch it if someone is already using it," and you would get destructive stomping between overlapping sensors and occupants. The state × action matrix is precisely what lets automation cooperate safely with whatever is already happening in the zone.

7.3 How inputs combine

- Multiple triggers targeting the same zone and event apply **independently** — there is no arbitration or priority; two sensors both saying "Set Occupied" simply both do it.
- One input instance can drive **several zones** at once (as the main-corridor sensor does), but a fixture belongs to exactly **one** zone.
- **Dim zone** and **Brighten zone** issue DALI group step-up / step-down broadcasts — fire-and-forget nudges that persist until the next state change. They suit Fixed Power and Scenes states, but fight the hunting loop in a Light Levels state.
- **Select next scene** cycles through the zone's defined scenes, wrapping at the end, independently of the zone state.

8 How the Zone Controls Its Fixtures

8.1 Group broadcast

When a state change or level command occurs, the zone drives its members through a single DALI group-broadcast to the group matching its zone number. This is why a zone acts as one entity: the fixtures move together, in step, from one command.

8.2 Fixture settings and inheritance

The low-level DALI behavior of the member fixtures — their **Minimum**, **Maximum** and **System Fail** arc levels, and their **Fade Time** and **Fade Rate** — is resolved through an inheritance chain: Fixture →Zone →Zone Type →Building default. The zone provides the middle tier. A fixture may keep its own individual settings, defer entirely to the zone, or mix the two on a per-property basis. The zone's Fixtures tab summarizes how many members currently defer to it and offers a one-click action to pull them all back into line.

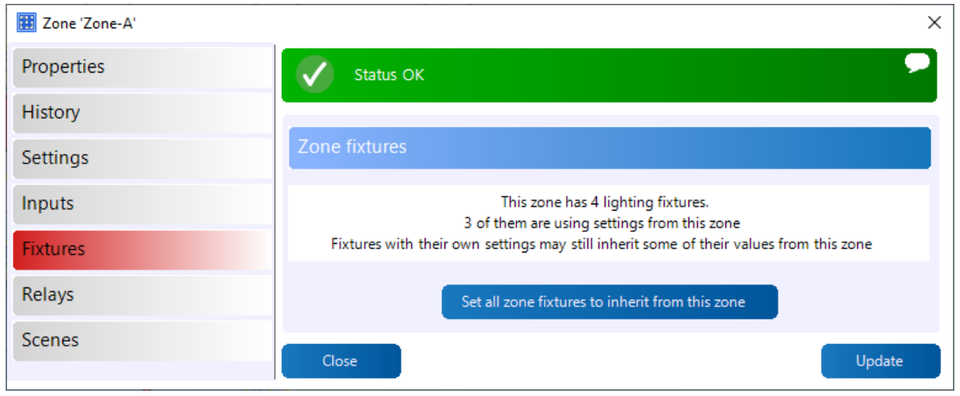


Figure 4. *A zone's default Fixture settings (Minimum, Maximum, System Fail, Fade Time, Fade Rate). Here every value is inherited (red) from the zone type — the zone passes them straight down to its member fixtures.*

> **Safety clamp**
>
> Whatever the chain resolves to, the fixture driver's own hardware minimum and maximum arc levels are the outer clamp. Zone, zone-type and building settings can only *narrow* within that hardware range — never widen it — and the System Fail level is the failsafe the fixture falls to if it loses DALI communications.

8.3 Scenes are per-fixture

A zone can define up to 16 scenes, referred to by number. The scene *list* lives at the zone — add and remove scenes there — but the *values* are stored per fixture: each member fixture holds its own arc level, or "No Change," for every scene. "No Change" means the fixture does nothing when that scene is selected, holding whatever level it already had. This is what lets one scene light part of a zone while leaving the rest untouched.

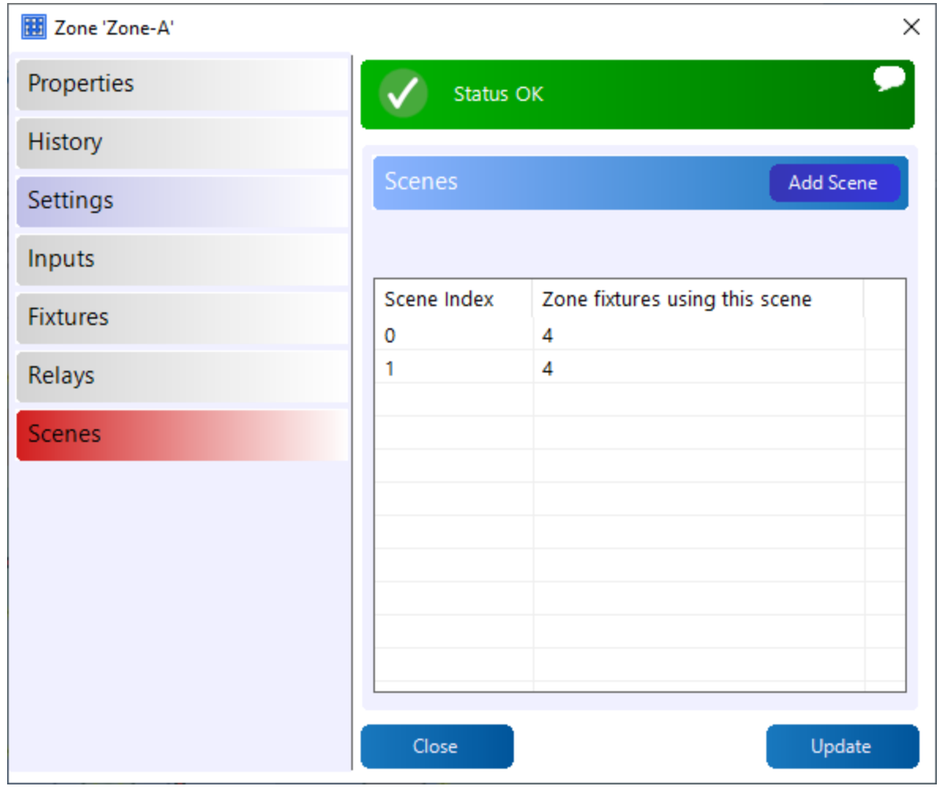


Figure 5. *The zone's Scenes list. Each scene is referenced by index (0–15), and the count shows how many member fixtures have an explicit level set for it — the rest are "No Change."*

9 How the Zone Controls Relays

Some zone members are not dimmable luminaires at all but **relays** — fixtures whose Assigned role is Relay, typically driving a switched load such as a plug circuit. Relays need special handling, and understanding why reveals a neat piece of the design.

A relay is usually a DALI device that maps **arc power to relay state**: zero means off, anything above zero means on. If such a relay were left in the zone's DALI group, then any state that produced arc power above zero would switch it on — which is rarely what you want, and it could never be controlled independently of the light level.

So the MCU deliberately keeps relay-role fixtures **out of the DALI group** and sends them separate, direct commands. Their on/off state is then set per zone-state by a dedicated matrix on the zone's Relays tab — completely independent of the lighting method, brightness or hunting in that state.

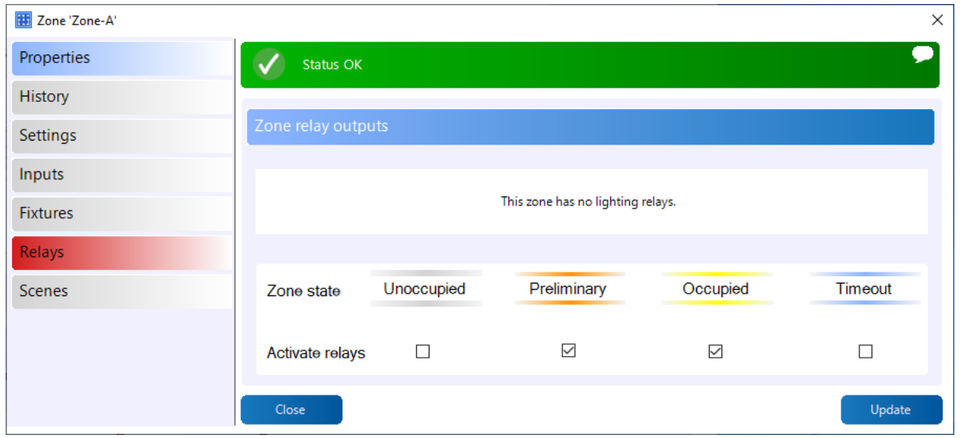


Figure 6. *The zone's Relays tab: a simple on/off choice per state. Here the relays are switched on only in Occupied and off in every other state.*

Because relays are commanded directly rather than through the group, they are unaffected by the state's lighting method and are not part of the scene system. A compliant relay also has nothing to act on when the zone issues Dim or Brighten group commands, so it is left switching cleanly on its own schedule of states.

10 Seeing the Zone in Action

10.1 Zone history

Each zone keeps a rolling record of its recent behavior for monitoring and troubleshooting: the state it has been in over time, the measured light level where a photosensor is present, and a breakdown of how much time it spent in each state. The view covers the last hour or the last 24 hours, and can be popped out into its own window to compare several zones side by side. This history is a diagnostic aid held for the last 24 hours only — it is separate from any long-term energy reporting.

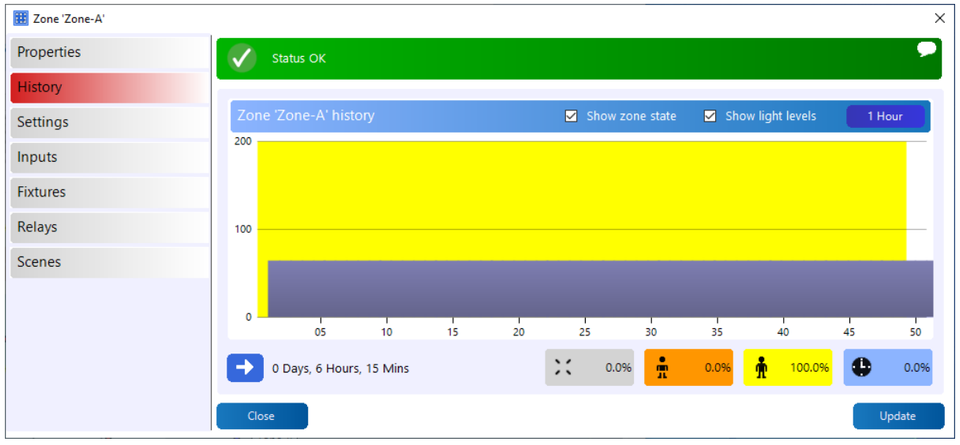


Figure 7. *Zone history: state bands over time with an optional light-level trace, plus a percentage breakdown of time spent in each of the four states.*

10.2 Demand Response as an overlay

Sitting above the state machine, **Demand Response** can trim a zone's output when the building is asked to shed load. It applies dynamically and multiplicatively on top of whatever the zone's current target happens to be — a fixed level, a scene level, or a light-level setpoint — without changing the logical state the zone is in. Occupant and manual control still work normally; only the resulting light output is reduced. Each zone (or zone type) defines what each of the three DR levels does: no action, dim by a percentage, or switch off.

11 The Flexible Heart

Pulling the model together: a zone is a single, autonomous control machine that groups fixtures for efficient DALI-group control, models occupancy through four purposeful states, lights each state by whichever of three strategies fits, transitions between states on configurable timeouts, and responds to sensors and switches through a state-aware action matrix — all while driving its dimmable fixtures and its switched relays each in the way that suits them.

Almost every worthwhile lighting behavior in the system is expressed by combining these few simple mechanisms. The corridor that lights your path and fades behind you, the office that daylight-harvests while you work and forgives you for sitting still, the meeting room that switches between scenes at the touch of a button — none of these needs bespoke logic. They are all just states, timeouts and a well-composed input matrix. That is what makes the zone the flexible heart of the NICOR MCU: a small set of clear ideas that recombine into an enormous range of real-world behavior.

PART I · INTRODUCTION & CORE CONCEPTS

5 Cross-Cutting Conventions

A few conventions recur across every part of the MCU's interface. Learning them once, here, means they need no re-explanation later: how the software signals status and warnings, how inherited settings are shown, the DALI-2 vocabulary the manual assumes, and how licensing gates features.

5.1 The warning and status model

Every object in the MCU — from a single fixture up to a whole building — reports its health through one consistent status control, shown as a banner at the top of its properties window and echoed by the object's icon on the floorplan. It has three states:

State	Appearance	Meaning
OK	Green — "Status OK"	No warnings; the object is healthy.
Warning	Red banner with a count (e.g. "2 / 3")	One or more warnings; the control cycles through them in turn.
Unavailable	Red — "Live status unavailable"	The gateway is disconnected, so no live status can be shown.

Two behaviors make this work across a large system. First, warnings **cascade**: a problem on a device is surfaced not only on that device but on every level above it, so a single fault deep in a network is visible at the network, controller and building levels without hunting for it. Second, where an object has more than one warning, the control **cycles** through them automatically, the counter showing which of how many is currently displayed.

5.2 The inheritance convention

Many settings in the MCU can either be set explicitly on an object or inherited from its parent — a fixture from its zone, a zone from its zone type, and so on up the chain. The software shows which is which using one convention everywhere:

↔**Inherited** — value shown in red with a red up-arrow pointing to the parent it comes from.

↔**Explicit** — a green left-arrow beside an editable field, set on this object itself.

Clicking the arrow toggles a setting between the two, and an **Inherit all** button resets every setting on a screen back to inherited at once. Wherever you see these red and green arrows — and you will see them on almost every properties window — this is what they mean.

5.3 A DALI-2 primer

The MCU manages lighting over **DALI-2** — the second edition of the Digital Addressable Lighting Interface (IEC 62386). DALI is a two-wire control bus connecting a controller to the lighting devices on it, and DALI-2 extends the original standard to cover **control devices** — the sensors and switches the MCU calls input devices — as well as the light fittings themselves. A handful of DALI-2 concepts appear throughout this manual; the table below is enough to follow them.

Term	What it means in the MCU
Short address	A device's address on the bus, 0–63 — hence up to 64 devices per network.
Group	One of 16 addressable groups (0–15). The MCU maps each zone to the group of the same number, so a single command drives the whole zone.

Term	What it means in the MCU
Arc power	The output (dimming) level, from 0 (off) to 254 (full). Usually shown as both the raw value and a percentage.
Scene	One of 16 stored levels (0–15) held per device, recalled together to produce a preset look.
Fade time / rate	How quickly a device moves to a new level — a smooth transition rather than an instant jump.
Instance	A single function within an input device — one button, or one sensor. A device can expose several.
System failure level	The level a device falls back to if it loses communication on the bus — a safe default.

Two points are worth drawing out.

Addressing is how the MCU's logical network model is mapped to actual physical network devices (fixtures and input devices). To be controlled, every device must have a unique short address, and much of Part III is concerned with making the MCU's model of which object sits at which address match the physical bus.

Instances are why one DALI-2 input device can do several things. DALI-2 input devices may provide multiple 'instances' of different types, each providing a separate control output. A multi-button wall switch presents each button as a separate instance, and a multi-sensor presents its occupancy and light-level functions as separate instances, each individual instance can be used by one or more zones on the same DALI network to drive zone behavior.

5.4 Licensing overview

Access to the MCU and its optional features is controlled by **licenses, which are always issued per client**. A license enables gateway connectivity for that client — without one, a client's gateways cannot connect — and further licenses unlock optional capabilities such as energy reporting and BACnet integration.

Licenses are issued and managed in the NICOR Administration App, and the mechanics are covered in Part IV. For now, the point is simply that some features described in this manual are optional and depend on the client's licensing; where a feature is license-gated, the manual says so.

PART II · FOR USERS

6 Getting Started

This chapter covers signing in to Site Manager, choosing which MCU server to connect to, and selecting a building to work in — the starting point for everything a user does day to day.

> **INSTALLING THE SITE MANAGER ONTO YOUR MACHINE**
>
> If Site Manager is not yet installed, you do not need a separate installer for it. Once the MCU service is running, open its installers page in a browser — `http://<host>:9000/web/installers` — and download Site Manager from there. The same page also carries the NICOR Administration App. Chapter 28 §28.3 describes the page in full; your administrator can tell you the host to use.

6.1 Logging in and choosing a server

Site Manager opens to a login window carrying the NICOR MCU branding, a **Username** and **Password** field (with a show/hide toggle beside the password), and a **Log In** button.

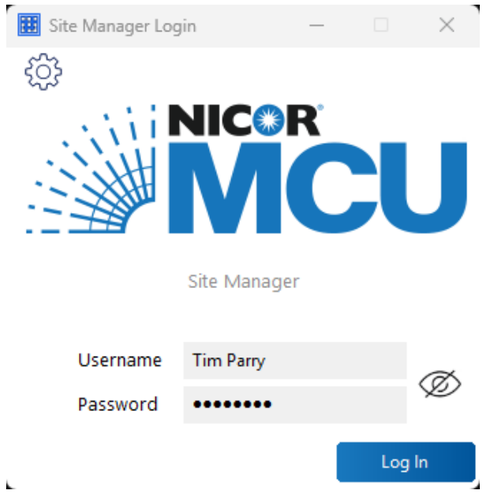


Figure 6.1. *The Site Manager login window.*

Before logging in, choose which MCU server to connect to. Click the **gear (settings) icon** at the top-left and open the **NICOR Lighting Servers** submenu. The list shows the servers available to connect to; the ticked entry is the current selection. Right-click an entry to edit or remove it, or choose **Add NICOR Lighting Server** to add a new one.

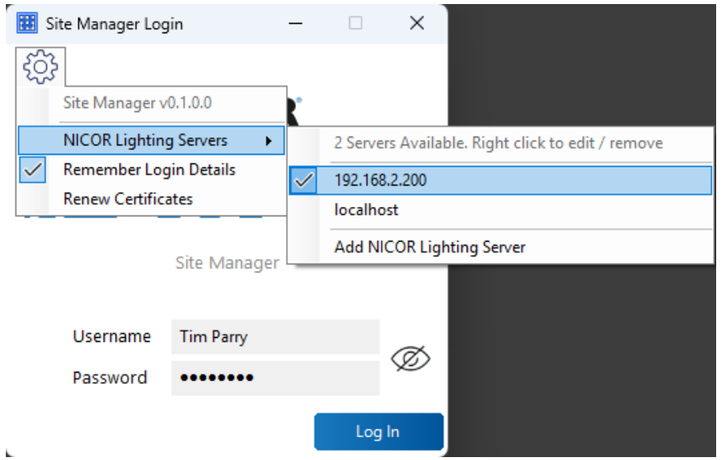


Figure 6.2. *The gear menu with the NICOR Lighting Servers list. Servers are identified by host; the ticked one is selected.*

Adding or editing a server opens the **MCU Server properties** dialog, which takes the server's **Host** and its **HTTP** and **HTTPS** ports. **Update** saves the entry.

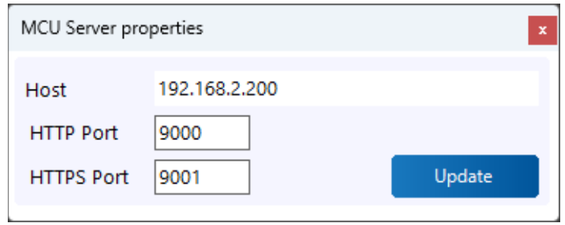


Figure 6.3. *The MCU Server properties dialog — host and port entry.*

> **Secured connections**
>
> Connections to an MCU server are secured with TLS, which is why servers are identified by host name. The first time Site Manager connects to a server it establishes trust with it; if that trust ever needs to be re-established, use **Renew Certificates** (next section). The full security model is covered in Part IV.

6.2 Remembering your login and renewing certificates

The gear menu carries two more options alongside the server list. **Remember Login Details**, when ticked, retains your username and password so you do not have to re-enter them next time. **Renew Certificates** re-establishes the secure trust between Site Manager and the MCU server — which is needed if a server's certificate has changed, for example after the server's certificate has been regenerated.

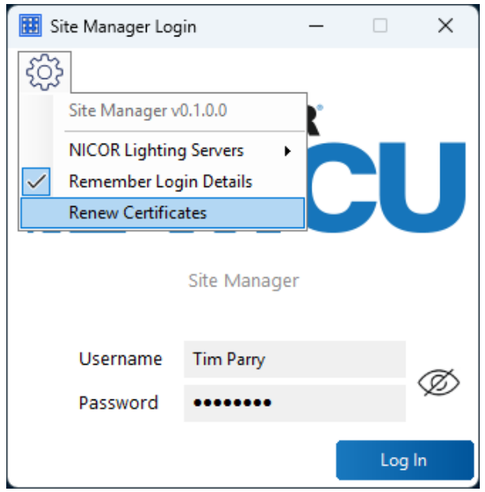


Figure 6.4. *The gear menu, showing Remember Login Details and Renew Certificates.*

6.3 Selecting a building

After you log in, the **Site Navigation** window appears. The left pane is a tree of everything you can access — your client, its locations, and the buildings within them. Selecting a location lists its buildings in the right pane; choose a building and click **Select building** (or double-click it) to open it.

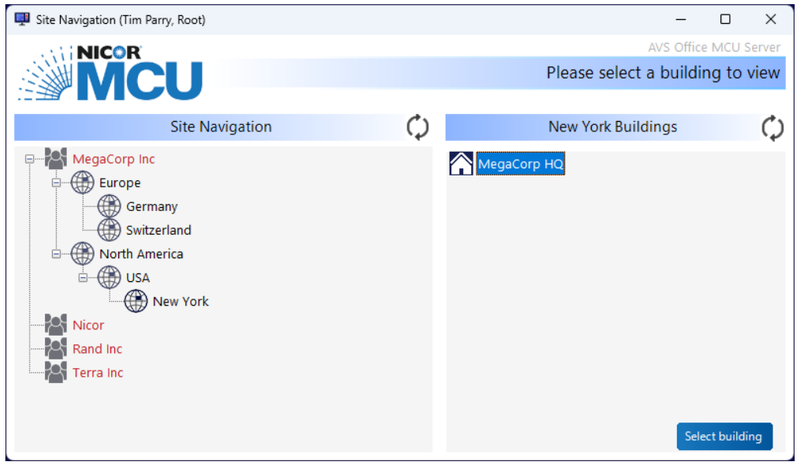


Figure 6.5. *The Site Navigation window: locations on the left, the selected location's buildings on the right.*

What you see here depends on your access. A standard user sees only their own client's locations and buildings; a user with wider access — such as the Root user shown above — sees multiple clients. This is the multi-tenant permission model from Part I in action: the same window shows each person exactly the part of the properties they are entitled to. Once a building opens, you arrive at the building view, described in the next chapter.

PART II · FOR USERS

7 The Building View

Opening a building brings you to the building view — the hub from which everything else is reached. This chapter explains what it shows, how to read its status at a glance, and where each part of the system is reached from.

7.1 Orientation

The building view is divided into two parts. A **navigation tree** on the left lists the building and the areas within it; selecting the building shows the building's own summary, while selecting an area shows that area's summary (covered in the next chapter). The larger pane on the right shows **status** for whatever is selected, together with a row of **action icons** for the things you can open from here.

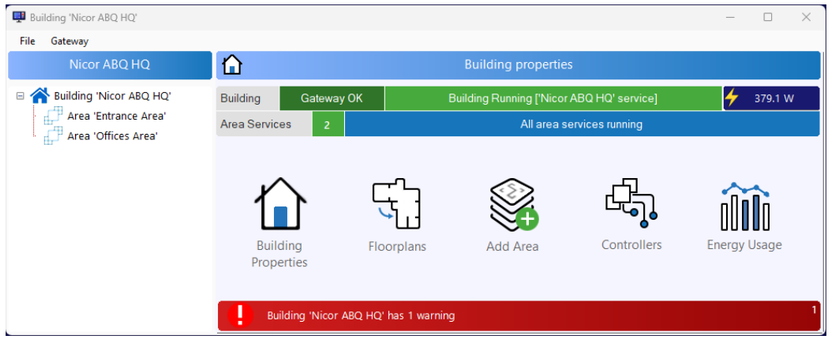


Figure 7.1. *The building view: navigation tree on the left, building status and action icons on the right, warnings along the bottom.*

The action icons across the center of the panel are the main routes into the rest of the system:

Icon	Opens
Building Properties	The building's settings — identity and location, gateway assignment, email notifications, zone types, default fixture settings and demand response.
Floorplans	The building's floorplan templates — the drawings that areas are laid out on.
Add Area	Creates a new area within the building.
Controllers	The building controller status window — the state and activity of every controller in the building (§7.4).
Energy Usage	The historical energy report for the building (see Chapter 10).

Which icons you see, and whether they do anything, depends on your permissions and the client's licensing — **Energy Usage** requires the energy license, and the configuration routes require the appropriate permission. If something described here is absent for you, that is why.

7.2 Reading the status bars

Two status rows sit above the icons and tell you, at a glance, whether the building is healthy:

- The **building row** shows the gateway connection state (for example **Gateway OK**), whether the building service is running, and the building's current **power draw in watts** — a live, estimated figure for all its lighting.
- The **area services row** shows how many areas the building has and whether all of their services are running.

Green means healthy. The power figure is worth a word of explanation: it is *estimated* rather than metered — calculated from what each fixture is currently doing and the power characteristics of its load type — and it is the same basis as the energy reporting in Chapter 10.

7.3 Warnings

Along the bottom of the view is the warning banner described in Chapter 5. When all is well it shows **Status OK** in green; when something needs attention it turns red and names the problem — for example "Building '…' has 1 warning" — with a counter on the right showing how many warnings there are in total. If there is more than one, the banner cycles through them.

> **Warnings cascade upwards**
>
> A warning shown at building level may well originate deep inside the building — on a controller, a network, or a single fixture. That is by design: problems are surfaced at every level above where they occur, so you can see that *something* is wrong without hunting for it. To find the source, follow the warnings down: select the area, then the object within it that is also showing a warning.

7.4 Controller status

The **Controllers** icon opens the **Building controller status** window: every IMS controller in the building on one screen, with its current state and what the gateway is doing with it. On a building with a handful of controllers the area summaries say as much; on one with dozens or hundreds, this is the quickest way to see whether they are all running and, if not, which ones are not.

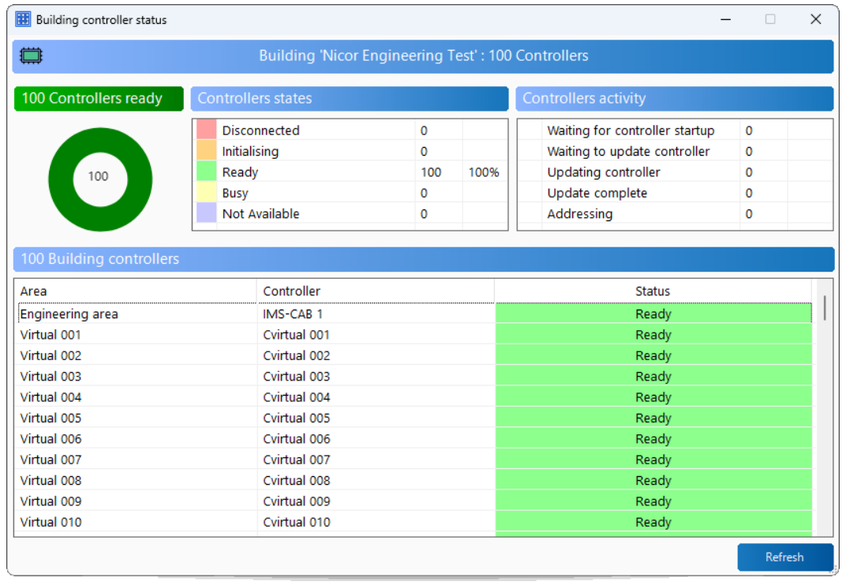


Figure 7.2. *The building controller status window for a building of 100 controllers, all of them ready.*

The top of the window summarizes the whole building. The headline and ring chart give the number of controllers that are **Ready**. **Controllers states** breaks the total down by state, with a count and percentage for each, in the colors used throughout the window; **Controllers activity** counts the controllers at each step of what the gateway is doing with them.

The states follow a controller from connection into service. While the gateway is bringing a controller up, it is **Initialising**, and the activity panel shows which of four steps it has reached: waiting for controller startup, waiting to update the controller, updating it, and update complete. It then normally becomes **Ready**. A controller carrying out a non-standard operation — addressing, for example — shows as **Busy** until the operation completes, and the operation is counted in the activity panel. **Disconnected** means the gateway has no connection to the controller. **Not Available** counts controllers that could not be queried or did not respond in time. It should always be zero; any other figure is worth investigating.

Beneath them, the list names every controller with its area and status, colored to match. **Refresh** brings the window up to date.

The window comes into its own while a building is starting up — after a gateway restart, or when a building is first brought online. The gateway works through the controllers, at a pace set by its controller initialization limit (Chapter 29), synchronizing each before it becomes Ready, and the window shows that progress across the whole building: how many are still initializing, how many are being updated, and how many are done. A controller that stays behind once the rest are Ready is the one to investigate, starting with its connection and network mode (Chapter 15).

7.5 The File and Gateway menus

Two menus sit at the top of the window. The **File** menu holds document-level actions for the building, and the **Gateway** menu holds actions relating to the gateway serving it — including the **gateway report**, a detailed log of what the gateway and its controllers have been doing. The report is a diagnostic tool rather than an everyday one; it is described in Part IV alongside the other monitoring facilities.

From here, the usual next step is to open an area and work with the floorplan itself — the subject of the next chapter.

PART II · FOR USERS

8 Working in an Area

The Area is where lighting actually happens, and its floorplan gives the clearest picture of what the system is doing. This chapter covers opening an area and reading its floorplan — the colors, the connections, and the live values shown on it.

8.1 Opening an area

Selecting an area in the building view's navigation tree shows that area's summary, laid out like the building summary one level down. The status rows show whether the area is **connected**, whether its service is running, and its current estimated **power draw**; a second row reports how many controller services the area has and whether all are running. Warnings appear along the bottom in the usual way.

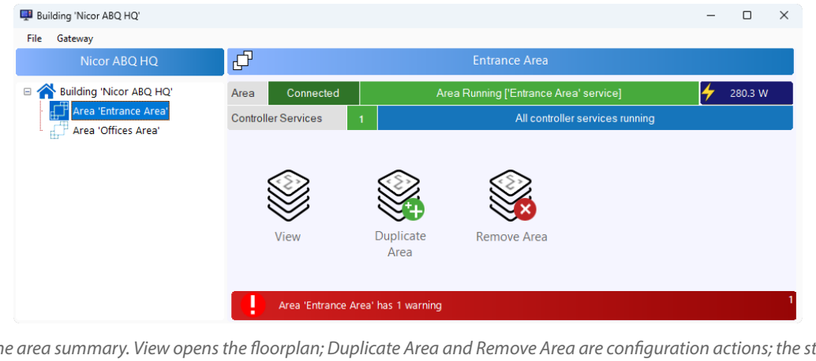


Figure 8.1. *The area summary. View opens the floorplan; Duplicate Area and Remove Area are configuration actions; the status rows and warning banner mirror the building view.*

Click **View** to open the area's floorplan. (The other two icons, Duplicate Area and Remove Area, are configuration actions and require the appropriate permission.)

Duplicate Area is for buildings laid out to a repeating pattern — floors or wings that share the same layout — where one area has already been designed and the next is the same again. After you confirm, the selected area is copied and saved in the building as a new area, complete with its schedule and floorplan sheets and with every controller, network, zone, fixture and input device reproduced exactly: settings, links and DALI addresses included. The copy takes the original's name followed by # and the next area number — duplicating Entrance Area in a building that already has three areas produces Entrance Area #4 — so the first thing to do is usually to rename it (Chapter 14 §14.7).

Some things are deliberately not carried across, because they belong to the original floor's hardware rather than its design. Each copied controller's IMS address is cleared, and its network mode is set to **Read only**, so the new area cannot reach or write to any hardware until it has been pointed at its own controllers. The product IDs of the copied fixtures and input devices are cleared too, since each identified one physical unit on the original floor. To bring the copy into service, give each controller the address of its own IMS, check that the copied model matches what is actually on the bus (Chapter 18), and only then set the controllers to Normal — the same sequence as bringing up any controller (Chapter 15 §15.10).

Because the DALI addresses are copied as they are, a duplicate fits best where the new floor was installed and addressed the same way as the original. Where its devices sit at different addresses, the copy is still a sound starting point, but the addresses must be resolved against the real devices before the area goes live. To reuse an area in a different building rather than this one, use Export →Area instead (Chapter 14 §14.9).

8.2 Reading the floorplan

The floorplan draws the area's objects over the building drawing. Everything on it carries meaning — shape identifies *what* an object is, color tells you *what state* it is in, and lines show *what is connected to what*.

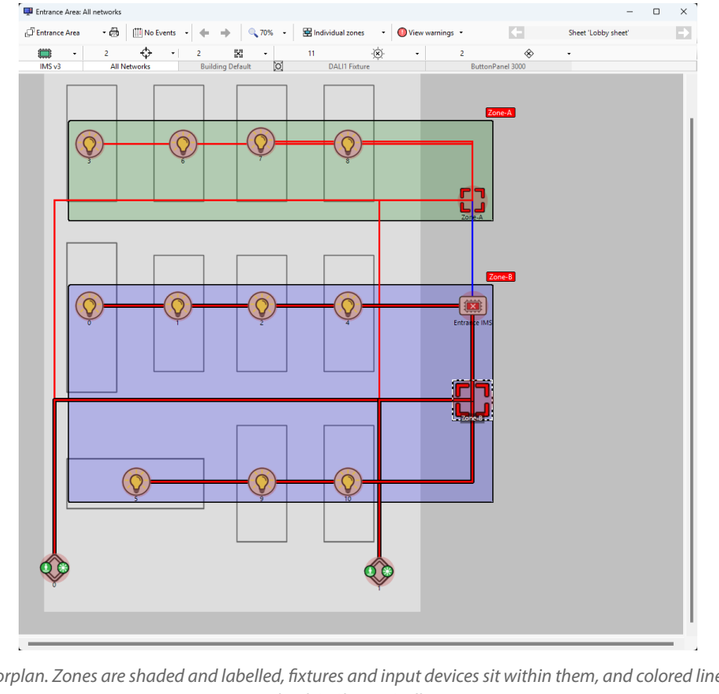


Figure 8.2. *An area floorplan. Zones are shaded and labelled, fixtures and input devices sit within them, and colored lines trace the connections back to the controller.*

On the floorplan	What it represents
Corner-bracket marker	A zone. Its color is the zone's current state — grey unoccupied, amber preliminary, yellow occupied, blue timing out, and red when the state is not known because the controller is not connected.
Shaded region and label	The extent of a zone, grouping the fixtures that belong to it, with the zone's name alongside.
Lamp icon	A fixture. The number beneath it is its DALI short address.
Diamond icon	An input device — a sensor or switch. The symbols on it show which functions it provides, such as occupancy and light level.
Controller icon	An IMS controller. Its own color and marking show its connection and synchronization state (Chapter 5).
Colored lines	Connections — tracing each device back to the network and controller that owns it.

Because zone color follows zone state, a glance across the floorplan tells you which parts of the space are in use, which are winding down, and which are at rest — the four-state model from Chapter 4, drawn out over the building.

8.3 Live values

Where a live connection to the gateway exists, the floorplan also shows current readings. Fixtures display their present **output as a percentage**, with a halo whose brightness reflects that output, so a dim area looks dim on the plan. Input devices with a light sensor display their measured **light level in lux** beside them.

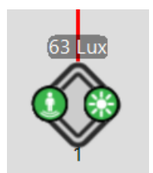


Figure 8.3. *An input device showing its live light-level reading.*

If no light sensor is present, no lux figure is shown. And when the gateway is disconnected, live values disappear altogether and the objects show the "Live status unavailable" state described in Chapter 5 — the configuration remains perfectly readable, but the system cannot report what is happening right now.

8.4 Hovering for detail

Hovering the pointer over any object brings up a tooltip summarizing it, which is the quickest way to answer, "what is this, and what is it doing?" without opening anything.

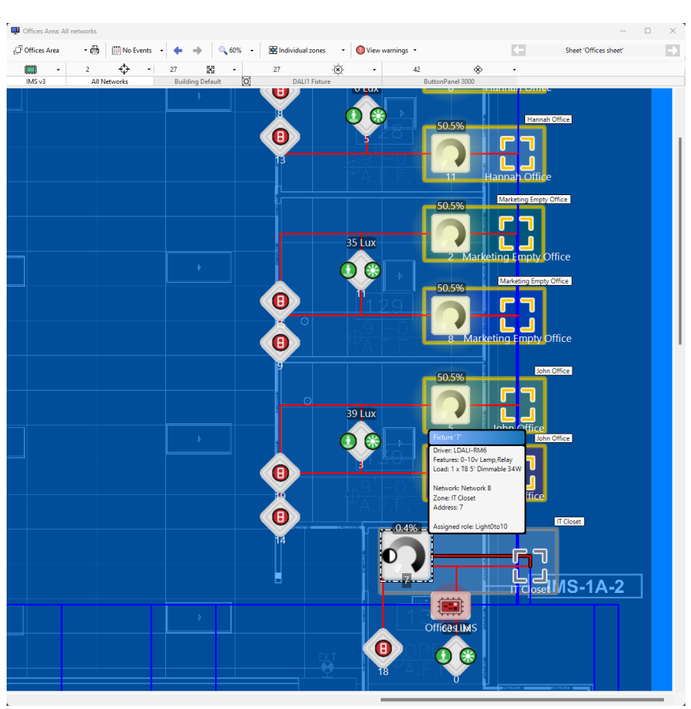


Figure 8.4. *Hovering a fixture summarizes its driver, load, network, zone, address and assigned role.*

The detail depends on the object:

- A **fixture** reports its driver type and features, the lamp load fitted to it, the network and zone it belongs to, its address, and its assigned role — whether it is acting as a dimmable light, a 0-10V light, or a relay.
- A **zone** reports its current state, its measured light level where a photosensor is present, and a count of the fixtures and relays it contains.
- An **input device** reports what it is and the instances it provides.

> **Looking closer**
>
> Tooltips are for a quick glance. To see or change an object's full settings, open its properties — covered in Part III. And if you have the **Modify Building** permission, a single click on a zone, network or input device opens a small popup for testing its behavior directly, which is the subject of the next chapter.

LIGHTING CONTROLS · MCU · USER GUIDE

PART II · FOR USERS

9 Manual Control

As well as watching what the system is doing, you can drive it directly from the floorplan. A single click on a zone, a network or an input device opens a small popup for exercising that object — invaluable when commissioning a space or tracking down why something is not behaving as expected.

> **Before you start — two conditions**
>
> These controls require the **Modify Building** permission; without it, the popups are unavailable. They also send commands to live hardware, so the controller must be **actively connected to the gateway** — they cannot be used while the connection is down.

9.1 Setting a zone's state

Clicking a zone opens a popup naming it, with one button for each of the four zone states — **Unoccupied, Preliminary, Occupied** and **Timeout**, each carrying its state icon. Clicking a button puts the zone into that state immediately.

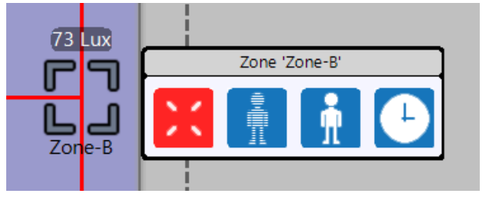


Figure 9.1. *The zone popup — one button per state.*

This is the fastest way to confirm that a zone is wired and configured as intended: set it to Occupied and check that the right lights turn on to the right level, then set it back to Unoccupied. Because you are setting the state rather than a raw level, everything downstream of the state applies — the configured lighting method, brightness, and any relays.

The zone continues to run normally afterwards. A state set by hand is simply the zone's current state, and the usual triggers — a timeout expiring, or a sensor reporting movement — will move it on from there as they normally would.

9.2 Broadcasting a level across a network

Clicking a network label opens a row of preset output levels — **Off, 25, 50, 75** and **100 %**. Choosing one broadcasts that level to **every fixture on the network at once**, ignoring zones entirely.

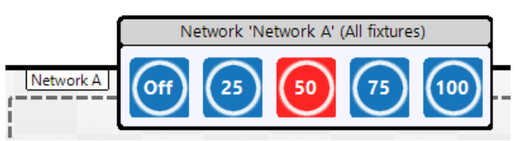


Figure 9.2. *The network broadcast — preset levels sent to every fixture on the network.*

This is a blunt instrument, and deliberately so. It is useful for proving that a whole network is alive and responding — flood everything to 100 % and look for the fixture that stays dark — without having to think about zone configuration.

The effect is **temporary**. Nothing is saved, and the fixtures return to normal control the moment their zone next issues a command — a state change, a sensor trigger or a scheduled event will all restore them.

> **Relays respond too**
>
> A broadcast reaches *every* fixture on the network, including any acting as relays. Because a relay switches on for any level above zero, broadcasting 25 %, 50 %, 75 % or 100 % will switch the network's relays **on**, and broadcasting Off will switch them **off** — regardless of what their zones would normally do. They return to normal control with everything else on the next zone command.

9.3 Triggering an input event

Clicking an input device lists the events it can produce — for a wall switch, the presses its buttons support; for a motion sensor, a detection. Clicking one fires that event exactly as though it had happened physically, so you can verify that an input produces the behavior you intended without walking to the switch or waving at the sensor.

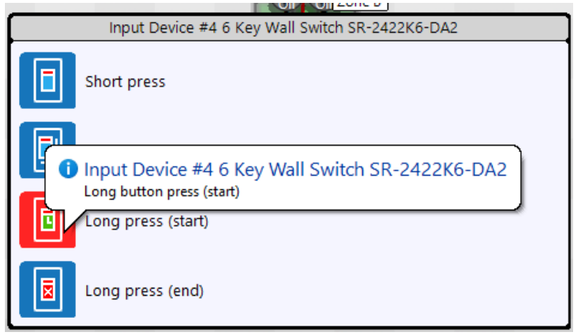


Figure 9.3. *Triggering an event on a wall switch. Only the events appropriate to the device are offered.*

Only events appropriate to the device appear, so the list is always meaningful. A button offers three kinds of press: short, double and long. Note that a long press is offered as two separate events, a **start** and an **end**: firing them in turn lets you test what happens while a button is held, and then on release — which mirrors how the physical device reports a long press.

This is the most precise of the three tools, because it exercises the actual chain the system will use in service: the event is delivered to every zone that listens for it, and each zone responds according to its own action matrix and its current state. If a zone does not react as you expect, that is exactly the mechanism to review — the same event can quite legitimately do different things in different zones, or nothing at all in a zone whose matrix specifies no action for its current state.

> **The same triggers, from the other direction**
>
> These events can also be fired from a zone's own Inputs list, described in Part III. The floorplan popup asks "what does this device do?"; the zone's Inputs list asks "what drives this zone?" Both fire the same events — use whichever matches the question you are asking.

PART II · FOR USERS

10 Energy Reporting

The energy report shows how much electricity a building's lighting has been using, and when. It answers both "how much did we use last month?" and "what does a typical day actually look like?" — and it can be narrowed to a single area or to one kind of space across the whole building.

> **A licensed feature, and an estimate**
>
> Energy reporting requires the **energy license**; without it the report is unavailable. Note also that the figures are **estimated, not metered**. The MCU calculates them from what each fixture was actually doing over time and the power characteristics recorded for its load type — so dimming, daylight harvesting and demand response are all reflected — but it is a calculation, not a reading from an electricity meter.

10.1 Opening the report

Open a building and click **Energy Usage**. The report window has its controls down the left-hand side and the chart filling the rest: a calendar to choose the starting date, two groups of buttons that choose what is plotted, and a tree that chooses what is measured.

10.2 Two ways to look at consumption

The button groups offer two quite different views. Despite being shown as two groups, they act as a **single choice** — picking any button deselects all the others, and the highlighted one tells you which view you are in.

View	Choices	Shows	Summary figure
Lighting load	1, 2, 5 or 7 days	Power draw in kW as it varied through the day — the shape of consumption over time.	**Average**
Energy usage	Week, Month or Year	Energy consumed in kWh, totaled per day (week and month) or per month (year).	**Total**

Use **lighting load** when you want to understand behavior — when the building comes to life in the morning, whether anything is running overnight, how much daylight harvesting is trimming the middle of the day.

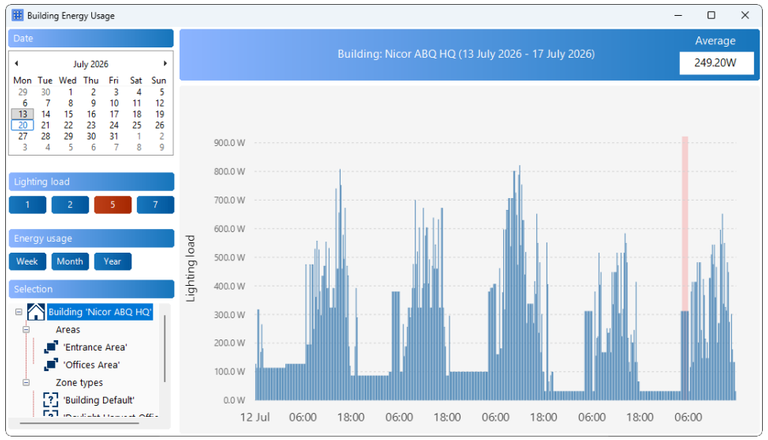


Figure 10.1. *Lighting load over five days. The fine detail shows the daily rhythm of the building, with the average across the period shown top-right.*

Use **energy usage** when you want quantities — comparing one day against another, one month against another, or totaling a period for reporting.

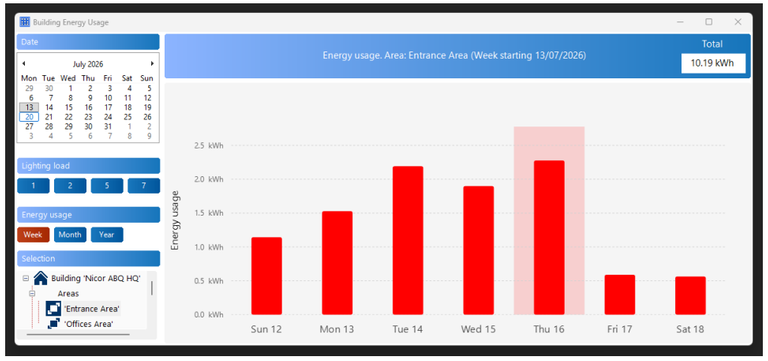


Figure 10.2. *Energy usage for a week, one bar per day, with the period total shown top-right.*

10.3 Choosing what to measure

The selection tree at the bottom-left sets the report's scope, at any of three levels:

- The **whole building** — everything, the default.
- A single **area** — useful for comparing one part of a building against another.
- A **zone type** — every zone of that type, in every area, across the whole building. This is the interesting one: it lets you ask questions like "how much are the offices using?" or "what do the corridors cost us overnight?" regardless of where they happen to be.

The chart is color-coded to the scope, so it is always obvious what you are looking at: **blue** for the whole building, **red** for an area, and **green** for a zone type. The heading above the chart names the scope and the period, and the summary figure top-right always corresponds to exactly what is plotted — change the scope or the period and it recalculates.

10.4 Gaps in the data

Occasionally a slot on the chart is drawn with a **pale red background**. This means the MCU did not have complete data for that period — typically because a controller or the gateway was not running at the time, or the system was undergoing maintenance.

The bar itself still shows whatever data *was* recorded, so a shaded slot is not necessarily zero — it is simply **incomplete, and known to be**. Treat it as a floor rather than a true figure. The shading appears at every level of aggregation, so a month that contains one incomplete day is itself flagged.

> **Where the numbers come from**
>
> Recent energy data is held in the live database, and older data is archived automatically into long-term storage — so a report spanning a long period draws on both without you needing to think about it. How far back the archive reaches depends on how the system has been configured; the retention model is covered in Part IV.

PART II · FOR USERS

11 What Your Permissions Allow

Not everyone sees the same MCU. If something described in this manual is missing or greyed out for you, it is almost certainly deliberate — this short chapter explains how access is decided, so you know what to ask for.

11.1 Two questions: what, and where

Your access is governed by two separate things, and it helps to keep them apart:

- Your **permissions** decide *what* you can do — whether you can operate a building, configure it, or administer the platform.
- Your **site access** decides *where* you can do it — the point in the hierarchy you are assigned to, which may be a client, a location, or a single building.

So two people can hold identical permissions and still see completely different buildings, and two people looking at the same building can have quite different abilities within it. Both are set by an administrator, and neither can be changed from within Site Manager.

The two interact in a way worth understanding. Site access places you at a point in the hierarchy; whether you can then *move down* from that point is governed by the **Site Navigation** permission — explained below.

11.2 The five permissions

Each user account carries a set of permissions, granted individually. In plain terms:

Permission	What it gives you
Site Navigation	The ability to browse downwards through the hierarchy from your assigned point — opening up the locations and buildings beneath it. Without this permission you are locked at the level you were assigned and cannot move deeper.
Site Management	Access to Site Manager itself — opening buildings, viewing floorplans, watching live status and reading reports. This is the everyday operating permission.
Modify Building	The ability to change things and to drive the lighting directly — including the quick-test popups described in Chapter 9. Without it, a building is effectively read-only to you.
Engineering	The commissioning and configuration work covered in Part III — controllers, networks, addressing, and the low-level engineering tools.
Administration	The platform administration covered in Part IV — managing users, clients, licensing and the backend services.

The permissions are cumulative rather than a ladder: an account is given exactly the ones it needs. A building manager might hold Site Navigation, Site Management and Modify Building; a commissioning engineer would add Engineering; a platform administrator would hold Administration as well.

11.3 Why something is greyed out

When you lack permission,— the affected controls are simply **greyed out or absent**. You can still see and understand the system; you just cannot act on that part of it. Common examples:

- You can open a building and watch it, but clicking a zone does nothing — you do not have **Modify Building**.

- You can view a floorplan, but the configuration and addressing options are unavailable — you do not have **Engineering**.
- A building you expected is not in the list — either it lies outside your **site access**, or it sits below your assigned point and you lack **Site Navigation** to browse down to it.

There is one other reason a feature may be missing that has nothing to do with your account: **licensing**. Energy reporting, for example, requires the energy license, and will be unavailable to everyone on an unlicensed system regardless of permissions. If a whole feature is missing for all users, licensing is the more likely explanation.

> **If you need more access**
>
> Permissions and site access are managed centrally by an administrator, using the tools described in Part IV. If you need access you do not have, that is who to ask — and it is worth saying which *task* you are trying to complete rather than which permission you think you need, since the right answer is sometimes a different one.

PART III · FOR ENGINEERS

12 Commissioning a New System

This chapter is the practical spine of Part III: how to take an MCU installation from nothing to a working, synchronized system. Later chapters cover each screen in detail; this one puts them in order.

12.1 Which path are you on?

Commissioning divides into two routes, and which one you take depends on a single question: **does a configured IMS installation already exist?**

	Path A — adoption	Path B — fresh build
Situation	The IMS controllers are already set up and running; the MCU is being added on top.	Both the IMS and the MCU are new. The controllers have addressed their devices automatically.
Who is authoritative	The IMS. The MCU is populated from it.	The MCU. The design is authored there.
Main work	Discovering and importing what already exists.	Building the floorplan, then binding it to the physical devices.

Both paths share the same preparation, described next.

12.2 Preparation (both paths)

Install the services

1. Install the **MCU service** (the DataManager) and initialize its database. The initial database ships with initial installer images for the **NICOR Administration App**, the **Gateway service** and **Site Manager**, so once the service is running its built-in installers page serves the other two — nothing else needs a separate installer. See Chapter 28 §28.3.

> **ONE INSTALL, THEN A WEB PAGE**
>
> The MCU service is the only component that must be installed the conventional way. With it running, open its installers page in a browser — `http://<host>:9000/web/installers` — and download the Administration App and Site Manager onto whichever machines need them.
>
> Standalone installers may still be supplied, but they are the exception. The web page needs nothing installed beyond the service that is already running.

Create the customer structure

2. In the NICOR Administration App, create the **client**, then its **locations**, then the **building** itself.

Apply the license

3. Issue a **gateway license** and add it to the client. This is easy to overlook and blocks everything that follows: **without a license the client's gateways cannot connect**. Licenses are always issued **per client**, and the same screen is where optional features such as energy reporting and BACnet are enabled.

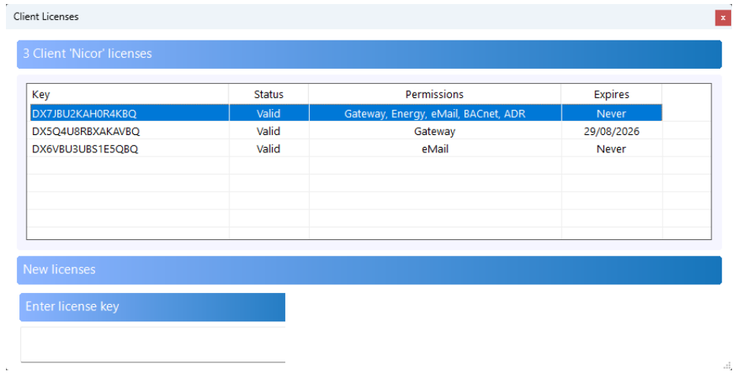


Figure 12.1. *The client's licenses. Each key carries the permissions it grants and an expiry; gateway connectivity depends on a valid one.*

Register the gateway and link the building

4. Register the gateway service, then open the building's properties and assign it on the **Gateway** tab. Use **Check connection** to confirm the two can talk. The first connection establishes the secure trust between them — the certificate model described in Part IV.

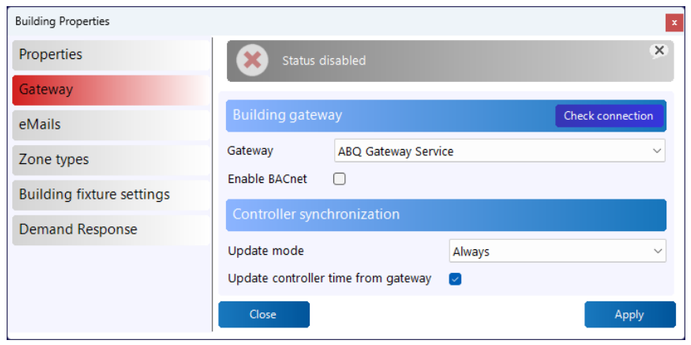


Figure 12.2. *Assigning a gateway to a building, with Check connection to verify it.*

12.3 Setting up the building

5. Add the **areas** the building divides into and import the **floorplan drawings** they will be laid out on.
6. Add the **controllers**. New controllers arrive in **Read-only** mode by default — leave them that way until commissioning is complete.

> **Why Read-only matters**
>
> Read-only lets the gateway connect to a controller and read its status, but never write to it. On an adoption project this is the safeguard that prevents an empty MCU from overwriting a working IMS configuration before you have imported it. Switching to **Normal** is deliberately the *last* step of commissioning, not an early one.

12.4 Path A — adopting an existing IMS

Here the controllers already hold everything: zones, fixtures, input devices and their settings. The task is to bring that model up into the MCU.

7. Run **Scan network for new devices** on each network. The MCU discovers everything the IMS holds and presents it for placing on the floorplan. Import **zones first** — the window enforces this — then fixtures and input devices, which arrive complete with their settings, zone links and addresses. Chapter 17 covers the mechanism in full.
8. If the controllers already hold schedules you want to keep, import the area schedule from them. This is a one-off onboarding step; from here on schedules are authored in the MCU.
9. Set the controllers to **Normal**. Commissioning is complete.

12.5 Path B — a fresh installation

Here the MCU is authoritative. A fresh IMS will have connected to its DALI networks and addressed the devices automatically, so the physical addresses already exist — what is missing is the MCU's model of what those devices *are* and how they should behave.

10. Author the design on the floorplan: create the **zones**, place the **fixtures** and **input devices**, and configure zone behavior — states, inputs, scenes and relays, as described in Chapter 15.
11. Bind the model to the hardware by resolving addresses. If the devices carry known **product IDs**, use **Update addresses from matching product IDs** to resolve a whole network in one pass. Otherwise use **Identify & Assign** to walk the devices physically, confirming each one on the plan as it identifies itself.
12. Author the **area schedules** in the MCU.
13. Set the controllers to **Normal**. The MCU's design is now pushed down and the system is live.

12.6 Combining the two

The paths are not mutually exclusive. A common and efficient pattern is to build the MCU floorplan **off-site** using generic placeholder devices — each identified only by a product ID or an address — and then, on site, run **Update controller from IMS** to resolve every address and import the real device details in a single pass. Chapter 17 describes this in full.

> **One rule to remember**
>
> "Update … from IMS" overwrites and prunes, but it can never *add*. If the IMS holds objects the MCU does not, the operation stops and directs you to scan for new devices instead. So the safe order is always **scan first, then update**.

12.7 Keeping it in step afterwards

Once live, the revision model described in Chapter 18 keeps the two sides synchronized automatically. Three situations call for deliberate action:

What happened	What to do
New devices added to a network	Address the new devices physically, then run Scan network for new devices and place them on the floorplan.
Zones, fixtures or input devices changed on the IMS	Run Update network from IMS to bring the changes up.
Devices physically re-addressed	Re-resolve with Update addresses from matching product IDs, or Identify & Assign where product IDs are not available.

Everything else — changes made in the MCU — needs no special handling. Save the area, and the gateway notices the new revision and updates the controllers automatically.

PART III · FOR ENGINEERS

13 Building Configuration

Settings made at building level reach everything inside it. This chapter covers the building's own properties — identity, gateway, notifications, and the defaults that zones and fixtures inherit — together with the floorplan drawings its areas are laid out on.

Building properties are reached from the **Building Properties** icon in the building view. The window has six pages, taken in turn below.

13.1 Identity and location

The first page carries the building's **identifier** — assigned by the system and used to reference the building elsewhere, such as in the gateway's list of assigned buildings — and its **name**, which is what everyone actually sees.

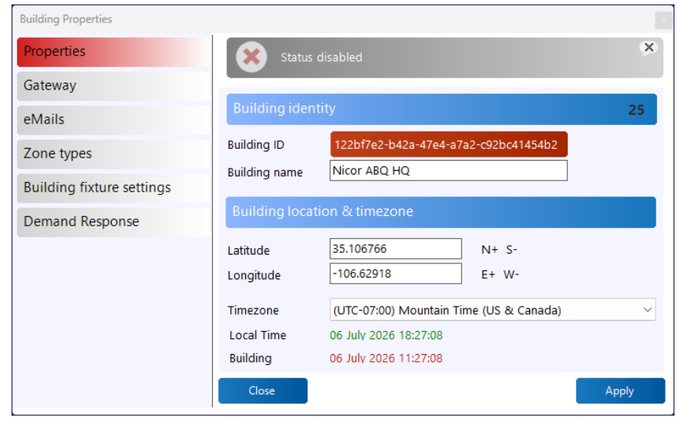


Figure 13.1. *Building identity, geographic position and time zone.*

Below that sit the building's **latitude and longitude** and its **time zone**. The controllers calculate their own local sunrise and sunset times from this position, which is what allows schedules to follow the sun through the year rather than a fixed clock time. The page shows both local and building time so that any discrepancy is obvious.

> **Get the position right**
>
> If the coordinates are wrong, sunrise and sunset events will fire at the wrong times — and because the controllers compute this themselves, the error will be consistent and easy to mistake for a scheduling fault. It is worth confirming the position when commissioning a building. Scheduling is covered in Chapter 20.

13.2 The gateway

The Gateway page assigns which gateway service serves this building, with a **Check connection** action to confirm the two can reach each other.

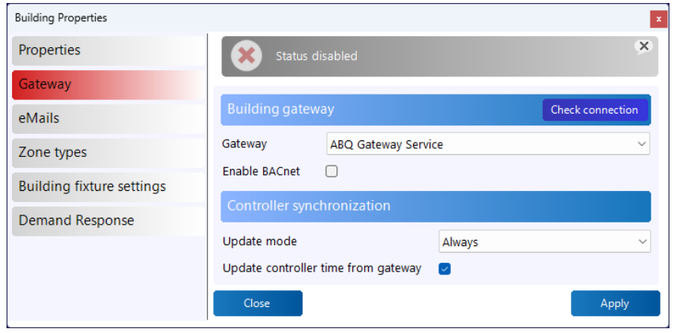


Figure 13.2. *Gateway assignment and the building-wide controller synchronization policy.*

It also carries two settings that shape synchronization across the whole building. **Update mode** is the default policy governing when the gateway pushes configuration down to controllers — the building-level half of the two-tier arrangement described in Chapter 19, which each controller can then restrict further. **Update controller time from gateway** keeps the controllers' clocks accurate, which matters for exactly the reason given above: they are doing their own sunrise and sunset calculations.

This page is also where **BACnet** is enabled for the building, where the client's license permits it.

13.3 Email notifications

The Email page sets who receives notifications about this building, and when. Two independent notifications are available — a **building summary** and a **building log** — each with its own recipient list and a **Send now** action for testing. A row of day buttons selects which days of the week they are sent on.

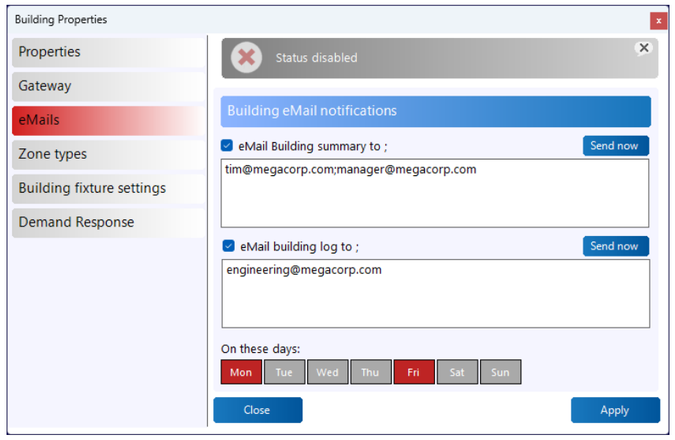


Figure 13.3. *Per-building email notifications, with recipients and sending days.*

These are the building's own notifications. The gateway can additionally send equivalent messages covering *all* the buildings it serves, configured separately in the Gateway Administration App and described in Part IV. The two operate independently, so it is worth checking both if recipients report duplicates or silence. Note: a valid Email license must be present for the buildings client to enable this feature.

13.4 Zone types

The Zone types page lists the templates available to this building's zones, showing how many zones are using each. Types are added and removed here, and double-clicking one opens it for editing.

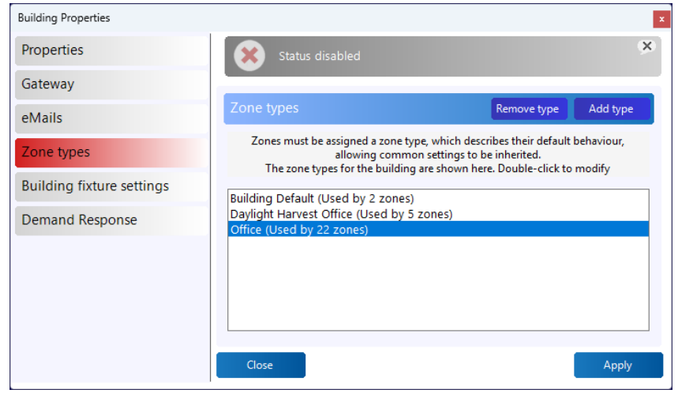


Figure 13.4. *The building's zone types, with a count of the zones using each.*

Because every zone must name a zone type, a building always has at least one — a default is created with the building so there is never an empty list. What the individual settings inside a zone type mean is covered in Chapter 16, alongside the zone settings they feed.

> **Design the types, not the zones**
>
> It is worth putting thought into zone types early. Defining "Office", "Corridor" and "Meeting Room" types properly means most zones need no individual configuration at all — and a later change to timings or levels can be made once, on the type, rather than zone by zone across the building.

13.5 Building fixture defaults

This page holds the root of the fixture-settings inheritance chain: the **minimum, maximum and system-failure levels** and the **fade time and rate** applied to every fixture in the building unless something below overrides them.

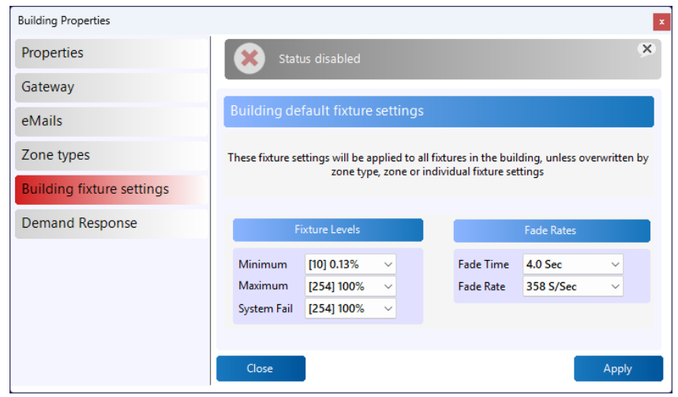


Figure 13.5. *Building-wide fixture defaults — the values everything else inherits from.*

The chain runs downward from here: building, then zone type, then zone, then the individual fixture, with each level able to override the one above it. Setting sensible values here means most fixtures never need touching individually.

13.6 Demand response

The Demand Response page shows the building's current demand-response state, whether an external request is present, and whether an override has been set — together with controls to drive it manually or return it to automatic.

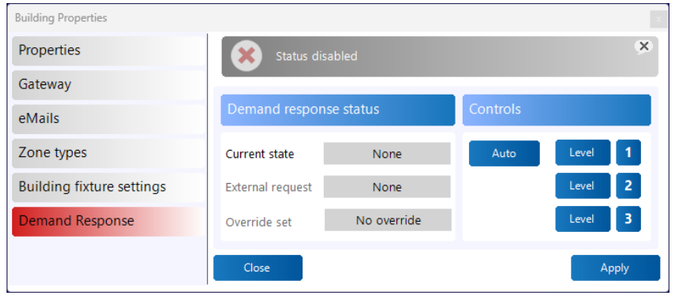


Figure 13.6. *Building demand-response status, with manual level controls and a return to automatic.*

This is the building-wide trigger; what each zone actually *does* at each level is configured per zone or zone type. Demand response is treated as a topic in its own right in Chapter 21.

13.7 ADR configuration

Where §13.6 covered demand response the building triggers for itself, the **ADR configuration** page is its external counterpart: it holds the settings that let the building receive demand-response signals from an outside utility or aggregator over the OpenADR v3.1 protocol. When these are configured and the building's demand response is in automatic mode, an incoming grid event drives the building's level exactly as a manual trigger would.

The page is present only for clients whose license includes ADR; without it the menu entry is disabled. Because the settings are numerous and connection-specific — VTN address, credentials, notification transport — they are documented in two places rather than repeated here: the behavior, and how external events sit alongside the building's own, in **Chapter 21 §21.8**; the field-by-field reference in **Appendix D**.

> **ONE LICENCE, ONE DIRECTION OF SIGNALLING**
>
> The ADR license governs only whether the building can act as an OpenADR client. It does not change how demand response behaves once a level is in force — that is the same machinery §13.6 drives manually, and the same per-zone-type reductions configured in Chapter 21. External demand response adds a new way for a level to be requested; it changes nothing about what the building does in response.

13.8 Floorplan drawings

Separately from its properties, a building holds the **floorplan drawings** that its areas are laid out on. These are managed from the **Floorplans** icon in the building view, which presents them as a thumbnail grid with actions to add, remove, export and edit them.

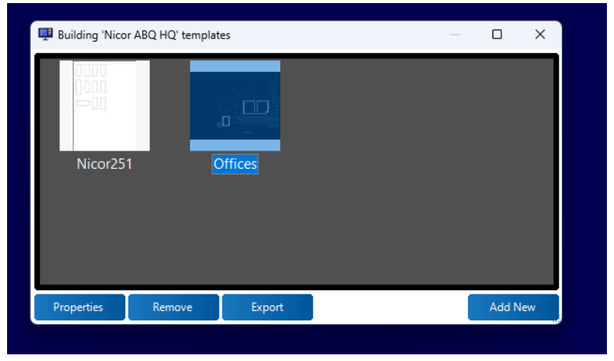


Figure 13.7. *The building's floorplan drawings.*

Opening a drawing's properties gives it a name and — importantly — controls how it will **appear behind the lighting objects**: **Invert** and **Grayscale** conversions, and explicit **foreground and background** colors.

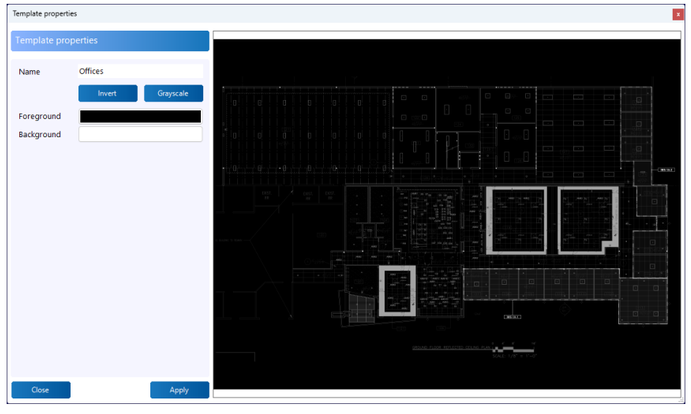


Figure 13.8. *A drawing's properties, with a live preview of how it will appear.*

These options exist because architectural drawings are usually black on white, which competes badly with the colored, glowing objects drawn on top of them. Inverting a drawing to white-on-dark, or dropping it to grayscale, makes the lighting layer far easier to read — which is why the floorplans shown throughout this manual appear as pale lines on a dark ground.

A drawing is shared building-wide: several areas, or several sheets within an area, can each use the same drawing and show a different part of it. How a sheet selects its portion is covered in Chapter 14.

PART III · FOR ENGINEERS

14 The Area Editor in Depth

The Area Editor is where an installation is actually built. Chapter 8 covered reading a floorplan; this chapter covers working on one — the menus and toolbars, how objects are created, arranged, copied and removed, and how the drawing behind them is set up.

14.1 The editor window

The editor puts two rows of controls above the canvas. The **upper row** acts on the area as a whole — saving, printing, scheduling, undo, zoom, display options and sheet selection. The **lower row** is the object toolbar, from which the objects themselves are created.

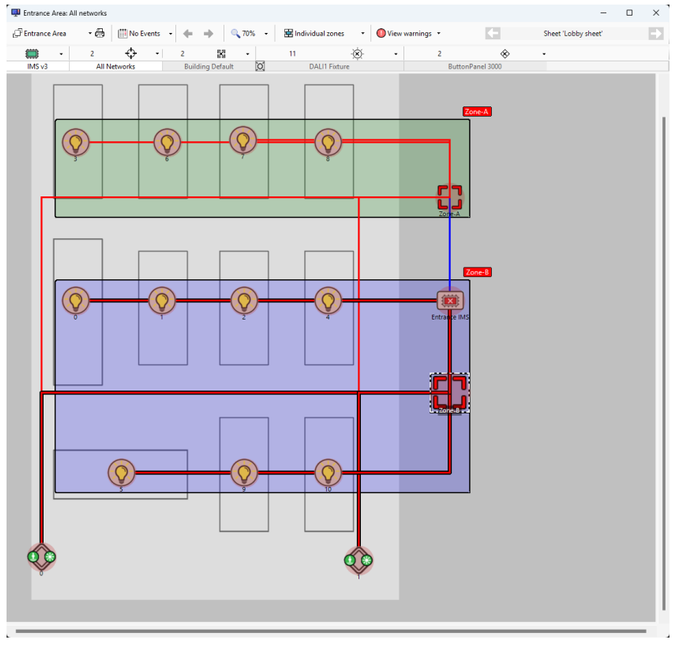


Figure 14.1. *The Area Editor. Area-level controls run along the top, the object toolbar sits beneath them, and the canvas fills the rest.*

Area actions

The area name at the top-left is a dropdown holding the area-level actions — opening its **properties**, **saving changes**, and **exporting**. Export now offers two forms, covered in §14.9: **Area**, which writes the area to a file for import into another building, and **Summary**, which produces a CSV sign-off sheet of everything the area contains.

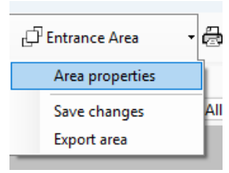


Figure 14.2. *The area dropdown.*

> **Saving is what reaches the controllers**
>
> Changes you make in the editor are local until the area is saved. Saving is what advances the area's revision and prompts the gateway to deliver the new configuration down to the controllers — the mechanism described in Chapter 19. Until then, nothing you have changed has reached the hardware.

Display options

The view dropdown controls how much the canvas shows and how it is oriented. Its submenus cover **Highlight** (what is emphasized), **Orientation** (rotating or flipping the plan to match how you are looking at the building), **Display options** (which details are drawn — live values, halos, labels), and **Edit options**.

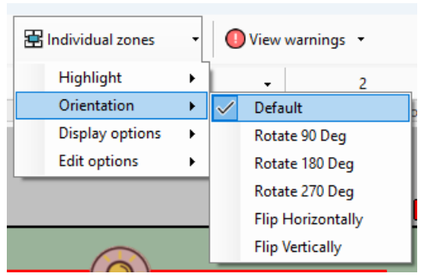


Figure 14.3. *Display options — here the Orientation submenu, which rotates or flips the plan.*

Alongside it are **undo and redo**, a **zoom** control, a **warnings** filter, and — at the right-hand end — the **sheet selector**. Where an area is drawn across several sheets, this names the sheet currently shown and steps forward and back through them. Only one sheet is displayed at a time, so the selector is how you move around a large area.

The object toolbar

The lower row is the object toolbar, and each entry does three jobs at once. It shows the **default type** that will be used when you drag that kind of object onto the plan, a **count** of how many the area already contains, and a dropdown that lets you change that default or navigate to existing objects.

The default types matter in practice. Before placing fixtures you choose which driver type they should be — and every fixture you then drop uses it until you change the default again. The same applies to controllers, to the zone type new zones inherit from, and to input devices. Setting the defaults to match the area you are about to build saves a great deal of correcting afterwards.

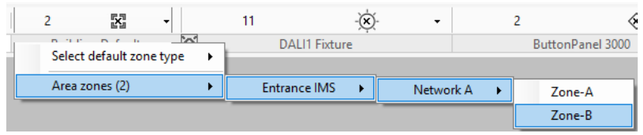


Figure 14.4. *An object toolbar dropdown. As well as setting the default type, it navigates down through controller and network to a specific zone.*

The working network

One toolbar entry deserves particular attention: the network selector, which sets the **working network**. Every fixture and input device must belong to a network, so when you drop one onto the plan the editor has to know which network to assign it to — and the working network is that answer.

Clicking the selector toggles through the choices in turn: **All Networks**, then each individual network, and back round again. Selecting a specific network changes the canvas in two visible ways:

- Objects belonging to the working network are drawn normally while everything else is **faded back**, so the network you are working on stands out clearly from the rest of the area.
- A **bounding box and label** are drawn around the extent of that network's objects, identifying at a glance which part of the floorplan it occupies.

With **All Networks** selected everything is shown at full strength, which is the right choice for reviewing or presenting an area. But because it names no single network, choosing a specific working network is a normal part of the build workflow — it is what tells the editor where new objects belong. It also determines where a copied object is pasted, as described below.

14.2 Creating and selecting objects

Objects are created by **dragging them from the toolbar onto the canvas**. Each arrives as the default type currently set for it, and fixtures and input devices are assigned to the current **working network** — so it is worth confirming both before you start placing. Placement is constrained by the DALI limits from Chapter 3 — sixteen zones, sixty-four fixtures and sixty-four input devices per network — and the editor simply will not let you exceed them.

To select:

- Click an object to select it.
- Hold **Shift** and click to add further objects to the selection.
- Or drag a **bounding box** around several objects at once.

Right-clicking an object opens a context menu appropriate to it — for a network label, for example, its properties, its parent, and the network update operations covered in Chapter 18.

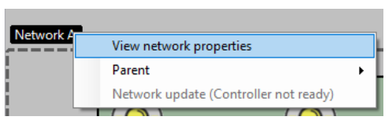


Figure 14.5. *A network's context menu. Operations unavailable in the current state are greyed out.*

14.3 Linking objects to zones

Placing a fixture on the floorplan puts it in the area, but does not yet say which zone controls it. Dragging is how that connection is made: **drop a fixture onto a zone and it becomes a member of that zone**, driven by it from then on.

The same gesture links inputs. Dropping an **input device instance** onto a zone tells that zone to listen to it — a motion sensor's detection, or a particular button on a wall switch. Because an input can serve any number of zones, dropping the same instance onto a second zone links it there too, without disturbing the first. This is the mechanism behind the corridor example in Chapter 4, where one sensor drives several zones at once.

> **Two ways to manage the same links**
>
> Drag-and-drop is the quickest way to build these relationships while laying out the plan. The same links can also be reviewed, added and removed from a zone's own properties — its Fixtures and Inputs pages, described in Chapter 16. Use whichever suits the moment: the floorplan when you are thinking spatially, the zone's properties when you are thinking about behavior.

Related objects need not sit on the same sheet. Where a link crosses sheets, the editor draws a small **link arrow** on each object, and the **direction of the arrow tells you which way to look**. An arrow *above* an object points to its parent — a fixture whose zone is on another sheet, or a zone whose controller is — and hovering over it names the sheet the parent sits on. An arrow *below* an object points to its children: a parent whose fixtures or zones live on other sheets carries one, and hovering lists those sheets. The relationship stays visible even when both ends cannot be shown at once, so a multi-sheet layout never hides how things are connected.

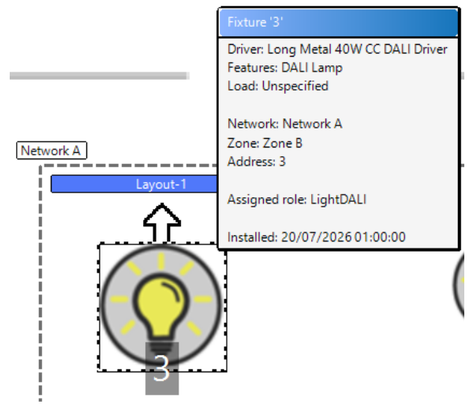


Figure 14.6. *An up arrow above a fixture: its parent zone is on another sheet. Hovering names that sheet — here Layout-1.*

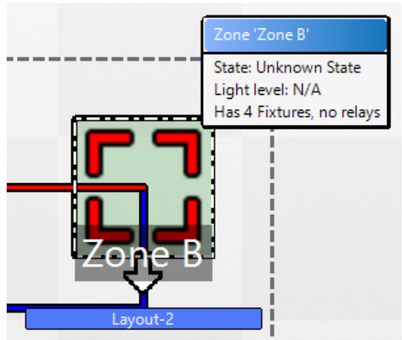


Figure 14.7. *A down arrow below a zone: it has children on another sheet. Hovering lists them — here on Layout-2.*

UP TO A PARENT, DOWN TO CHILDREN

The rule is worth committing to memory, because it turns a crowded multi-sheet area into something readable at a glance. **Up** means "what controls me is elsewhere"; **down** means "what I control is elsewhere." A fixture separated from its zone shows an up arrow; the zone shows a down arrow in return, so the same split is visible from both sides.

14.4 Copying and pasting

Right-clicking a selection offers to **copy** it; right-clicking empty canvas then offers the paste options appropriate to what was copied. An ordinary paste duplicates the objects themselves, keeping their original properties — a duplicated fixture, for instance, still belongs to the same zone as the original.

Two cases go further. When exactly one **zone** or one **controller** is selected, a **complete copy** is offered, which duplicates the whole hierarchy beneath it — a zone with all its fixtures and the input devices it uses, or a controller with all its networks, zones, fixtures and input devices. This is the quickest way to replicate a floor of identical offices.

Point to note	Behavior
Addresses are cleared	Copied devices arrive with their short addresses unassigned — two objects cannot share a physical address. Resolve them afterwards (Chapter 18).
Limits are respected	A paste that would exceed a network's capacity is clipped to fit rather than failing outright.
Pasting to another network	The paste goes to the current working network. To copy a zone onto a different network, change the working network first — the paste option then names the destination explicitly.

14.5 Moving objects

An object is moved on the sheet by dragging it. Several can be moved together: click one and shift- or ctrl-click others to build a selection, or drag a box around them, then drag any member and the whole selection travels with it, keeping its relative arrangement. Nothing about a plain drag changes what an object *belongs to* — it only changes where it sits on the sheet.

Holding **Ctrl** while dragging extends the move to an object's children. Ctrl-dragging a **zone** brings its fixtures with it. Ctrl-dragging a **controller** brings **everything beneath it** — every zone, fixture and input device on all of its networks — so the whole installation moves as one.

Dragging only moves objects within the current sheet. To move a selection to a **different sheet** of the same area, right-click it and choose **Move to sheet**, then the destination — the object keeps all its properties and its links, and appears on the sheet you named. The same can be done by cutting from one sheet and pasting onto another (§14.4).

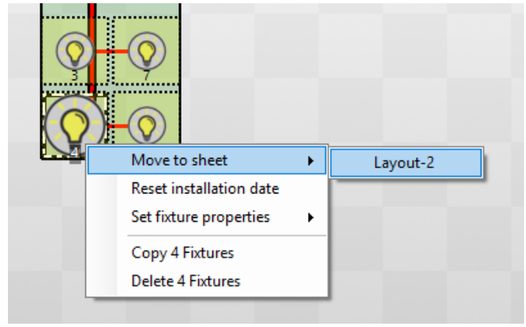


Figure 14.8. *Move to sheet on the object context menu, listing the other sheets in the area as destinations.*

MOVING A SHEET DOES NOT MOVE THE LINK

A fixture and the zone it belongs to can sit on different sheets — the editor draws a link arrow on each when they do (§14.3). So moving a fixture to another sheet does not detach it from its zone; it becomes one of those cross-sheet links, an arrow appearing at both ends. The relationship is unchanged, only its drawing.

Moving an object between *sheets* is a spatial operation — it changes where the object is drawn, not what it belongs to. Changing what it belongs to — re-zoning a fixture, or moving it to another network — is a separate action, **Set parent**, on the same context menu. Because that reaches the hardware and carries consequences, it is covered with the objects themselves in Chapter 17 §17.9.

Input devices stay put — and that is deliberate

Ctrl-dragging a zone moves its fixtures but *not* its input devices, because a single sensor or switch may serve several zones. Moving it with one zone would drag it away from the others. This is the owned-versus-shared distinction from Chapter 3 showing up in the interface: fixtures belong to exactly one zone and travel with it; input devices are shared and stay where they are.

A controller is different because nothing is left behind to be shared *with*. Ctrl-dragging a controller takes its input devices too, since every zone that could be listening to them is moving in the same gesture — the shared-versus-owned tension only arises when part of the model stays put.

14.6 Deleting objects

Deletion follows the same ownership logic:

- Deleting a **controller** removes everything beneath it — its networks and all their contents.
- Deleting a **zone** removes the fixtures that belong to it, but leaves its input devices in place; delete those separately if they are genuinely no longer needed.
- Networks **cannot** be deleted on their own — they are physical children of their controller and exist for as long as it does.

There is no confirmation prompt, because a **single undo restores the entire cascade** — the whole deleted subtree comes back with its settings intact. The editor keeps a generous undo history, so recovering from a mis-click is routine.

> **Deleting does not touch the hardware directly**
>
> Removing objects changes the MCU's model, not the physical bus. The devices carry on until the change reaches them the normal way: save the area, the gateway notices the new revision, and the controllers reconcile to match.

14.7 Area properties and sheets

The area's own properties are reached from the area dropdown. The **Properties** tab carries the area name and the **fixture halo scale** — how large the brightness halos around fixtures are drawn, which is purely a display preference.

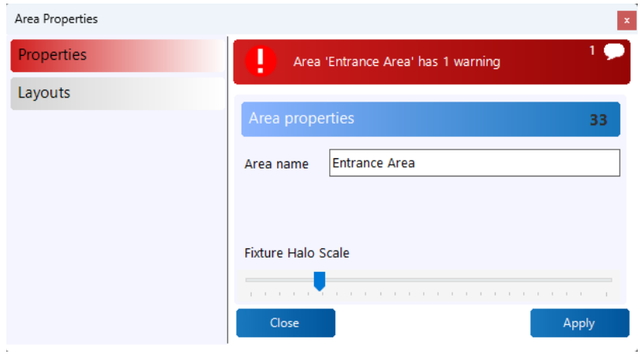


Figure 14.9. *Area properties — name and halo scale.*

The **Layouts** tab is more substantial. An area can be drawn across several **sheets** — useful for a large floor, or for separating a building into levels — and each sheet has its own settings: which floorplan drawing it uses, whether that drawing is cropped, the scale of icons and text, the opacity of the drawing behind them, foreground and background colors, and the real-world **dimensions** of the space in meters.

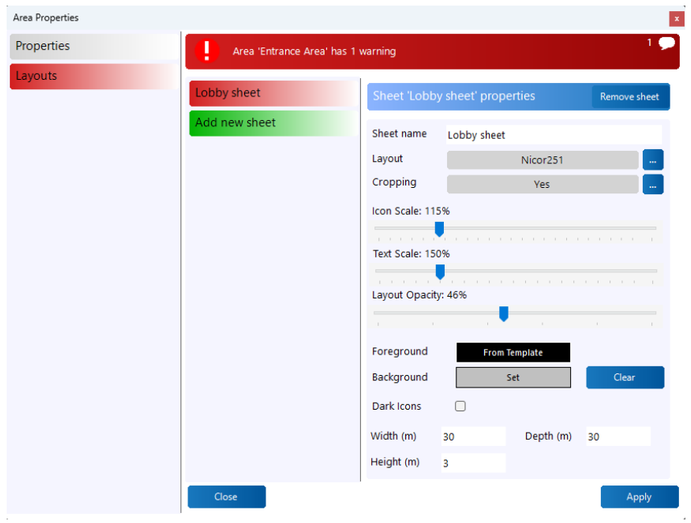


Figure 14.10. *The Layouts tab: the area's sheets on the left, the selected sheet's settings on the right.*

Objects are free to sit on whichever sheet suits them, and links between them are unaffected by the split — a zone on one

sheet may perfectly well control fixtures drawn on another, with the link arrows described in section 14.3 showing where the counterparts live. Sheets are a way of dividing up the *drawing*, not the *system*.

Cropping opens a dedicated view where you drag a rectangle over the drawing to select the portion this sheet should show — the practical way to use one large architectural drawing across several sheets, each showing its own part. A **No clipping** option restores the full drawing.

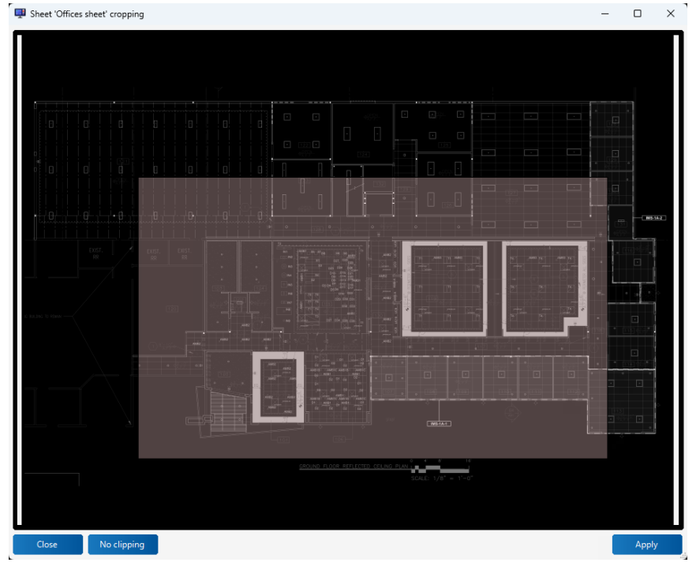


Figure 14.11. *Cropping a sheet — the highlighted rectangle is the portion this sheet will show.*

14.8 Printing a sheet

The print icon produces a paper copy of the **current sheet**. Choose portrait or landscape and a scale, and the dialog shows how many pages the sheet will tile across — with a live preview of the breakdown, so you can see the paper cost before committing. **Print** then hands over to the standard Windows print dialog.

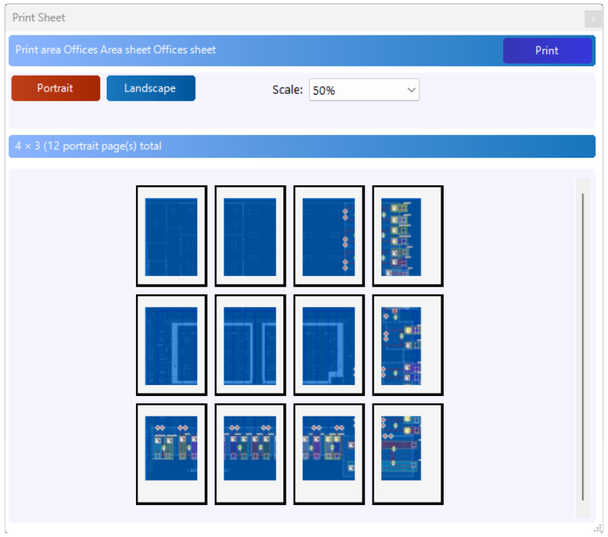


Figure 14.12. *The print dialog, showing how the sheet tiles across pages at the chosen scale.*

Two things are worth knowing. The printout reflects the **display options currently set** — so set the view up the way you want it before printing. And printing is **per sheet**; a multi-sheet area is printed one sheet at a time. Every tile in the grid is produced, including largely empty edge pages, and each page carries a header identifying the area and sheet so a tiled set stays collatable.

14.9 Exporting an area

The **Export** item on the area dropdown has two subitems, and they serve quite different purposes.

Export →Area writes the whole area — its sheets, zones, fixtures, input devices and their configuration — to a file that can be imported into another building. It is the tool for reusing a design: a standard floor laid out once and replicated, or a template carried between sites.

Export →Summary produces something different — a **CSV sign-off sheet** of everything in the area, laid out for opening in Excel or any spreadsheet. It is not a backup and cannot be imported back; it is a record, for commissioning handover, asset registers, or checking an installation against the design on paper.

	A	B	C	D	E	F	G	H	I	J
1	Object	Type	Details	Network	Address	Sheet	X	Y	Installed	Comment
2	Controller	IMS v3	Offices IMS		http://192.168.2.250	Offices sheet	86.5	36.7		
3	Zone	Office	Issac Office	Network A	7	Offices sheet	48.2	88.7		
4	Zone	Daylight Harvest Office	HR Storage	Network A	4	Offices sheet	31	88.7		
5	Zone	Office	Issac Office	Network A	6	Offices sheet	42.3	88.7		
6	Zone	Daylight Harvest Office	Small Meeting Room	Network A	1	Offices sheet	11.3	88.7		
7	Zone	Office	Office	Network A	9	Offices sheet	60.1	88.7		
8	Zone	Office	Office	Network A	8	Offices sheet	54.2	88.7		
9	Zone	Daylight Harvest Office	Small Meeting Room	Network A	0	Offices sheet	6	88.7		
10	Zone	Daylight Harvest Office	HR Storage	Network A	5	Offices sheet	36.7	88.7		
11	Zone	Office	Trevor Office	Network A	11	Offices sheet	70.6	88.7		
12	Zone	Office	Kiva 12 PL	Network A	12	Offices sheet	94.7	87.6		
13	Zone	Office	HR Office	Network A	2	Offices sheet	18.8	88.7		
14	Zone	Office	HR Office	Network A	3	Offices sheet	25	88.7		
15	Zone	Office	Trevor Office	Network A	10	Offices sheet	66.2	88.7		
16	Zone	Office	Kiva 13 LL	Network A	13	Offices sheet	77.1	83.8		
17	Zone	Daylight Harvest Office	Kiva Daylight harvest	Network A	14	Offices sheet	77.1	94.2		
18	Fixture	LDALI-RM6	1 T8 5' Dimmable 34W	Network A	12	Offices sheet	66.2	81.1		
19	Fixture	LDALI-RM6	1 T8 5' Dimmable 34W	Network A	4	Offices sheet	48.1	81.1		
20	Fixture	LDALI-RM6	1 T8 5' Dimmable 34W	Network A	3	Offices sheet	36.7	81.1		
21	Fixture	LDALI-RM6	2 T8 5' Dimmable 28W	Network A	14	Offices sheet	91	94.2	24/07/2026	2 Troffers
22	Fixture	LDALI-RM6	1 T8 5' Dimmable 34W	Network A	10	Offices sheet	70.5	81.1		
23	Fixture	LDALI-RM6	1 T8 5' Dimmable 34W	Network A	5	Offices sheet	18.8	81.2		
24	Fixture	LDALI-RM6	1 T8 5' Dimmable 34W	Network A	7	Offices sheet	12.1	81.9		
25	Fixture	LDALI-RM6	1 T8 5' Dimmable 34W	Network A	13	Offices sheet	81.9	83.8		2 Troffers

Figure 14.13. *A summary export opened in Excel. One row per object — controllers, zones, fixtures and input devices — across the whole area.*

Each row is one object, and the **Object** column says which kind. The remaining columns hold the same fields for every row, but a few carry different information depending on the object type — which is worth understanding before reading the sheet.

Column	What it holds
Object	The kind of object: Controller, Zone, Fixture or Input device.
Type	The object's type. This is the zone type for a zone ("Office", "Daylight Harvest Office"), the driver model for a fixture, and the IMS type for a controller — currently always IMS v3, the only type supported at present.
Details	The object's name, or for a fixture its driver-load description ("1 T8 5' Dimmable 34W").
Network	The network the object belongs to.
Address	Depends on the row. For a **controller** it is the controller's network address; for a **zone** the zone number (0–15); for a **fixture or input device** the DALI short address.
Sheet	The floorplan sheet the object sits on.
X, Y	The object's position on the floorplan, each given as a percentage of the sheet's width and height. A locating aid for finding a device on the plan — *not* a coordinate or an address.
Installed	The installation date, where one has been recorded against the object.
Comment	The object's free-text comment, where set.

> **A RECORD, NOT A BACKUP**
>
> The summary is a snapshot for people, not for the system. It cannot be imported, and it captures the area as it stands at the moment of export — so regenerate it after configuration changes rather than relying on an older copy. For moving a design between buildings, use Export →Area instead.

PART III · FOR ENGINEERS

15 Controllers and Networks

A controller is where the MCU's model of a building stops being a drawing and starts being hardware. This chapter covers the controller object and the five tabs of its properties window, the DALI networks that hang beneath it, and the pair of settings — the building's update mode and the controller's network mode — that between them decide whether anything you configure is permitted to reach the bus at all.

15.1 Where a controller sits

The hierarchy introduced in Chapter 3 runs Client → Location → Building → Area → Controller → Network → Zone → fixtures and input devices. The controller is the hinge in that chain. Everything above it is organizational — a way of saying who owns what and where it is. Everything below it is electrical.

A controller is a physical IMS unit sitting on the site network, and it is added to an *area* (Chapter 14). An area may hold several. Each controller carries one or more DALI networks, one per DALI channel it provides, and each network is a bus with its own devices and its own zones.

It is also the boundary of autonomy. The IMS runs the zone logic itself, locally and continuously: occupancy is evaluated, timeouts count down and lights respond whether or not the gateway is reachable, whether or not the DataManager is running, and whether or not anyone is logged in. Chapter 2 sets out what that means for the system as a whole. The practical outcome for this chapter is that configuring a controller is not the same as controlling it — you are handing it instructions it will then carry out on its own.

15.2 The controller's properties

Selecting a controller and opening its properties gives a window with five tabs down the left. The first holds the settings that identify the unit and govern how the MCU is allowed to talk to it.

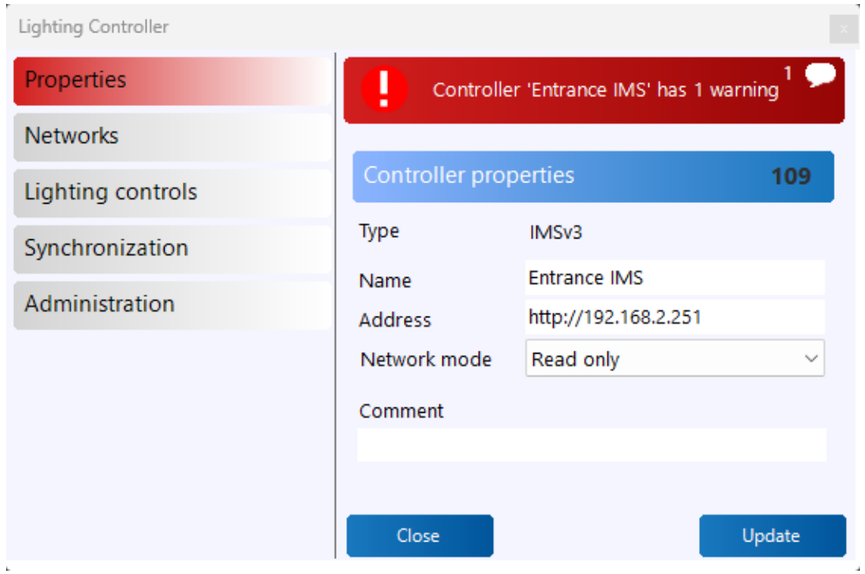


Figure 15.1. *The controller's Properties tab. Type is reported by the hardware; Name, Address, Network mode and Comment are yours to set.*

- **Type** — the controller model, as reported by the unit itself. It is not editable: it is whatever the hardware says it is, and it tells you which generation of IMS you are dealing with.
- **Name** — free text, and the label the controller carries everywhere else in the software. Name controllers for *where they are* rather than what they are. "Entrance IMS" tells an engineer standing in a plant room something useful; "Controller 2" does not.
- **Address** — where the controller is reached on the site network. An address that is wrong, or that has changed because the unit was given a different lease, is the single most common reason a newly added controller never comes up.
- **Network mode** — whether, and in which direction, configuration is allowed to move between the MCU and this controller. It is important enough to have its own section: see §15.5.
- **Comment** — free text. Worth using for the facts of the hardware rather than of the configuration: physical location, which circuit feeds it, and installation date.

The number at the right of the panel header is the controller's data revision. Revisions are the mechanism by which the MCU and the controller work out whether they are in step and, if not, which of them is ahead; Chapter 19 covers how they are used. The status banner across the top follows the conventions in Chapter 5 §5.1, and because warnings cascade upward, a fault on a single device on a single network will be visible here too.

15.3 The networks on a controller

The Networks tab lists the DALI networks defined on this controller.

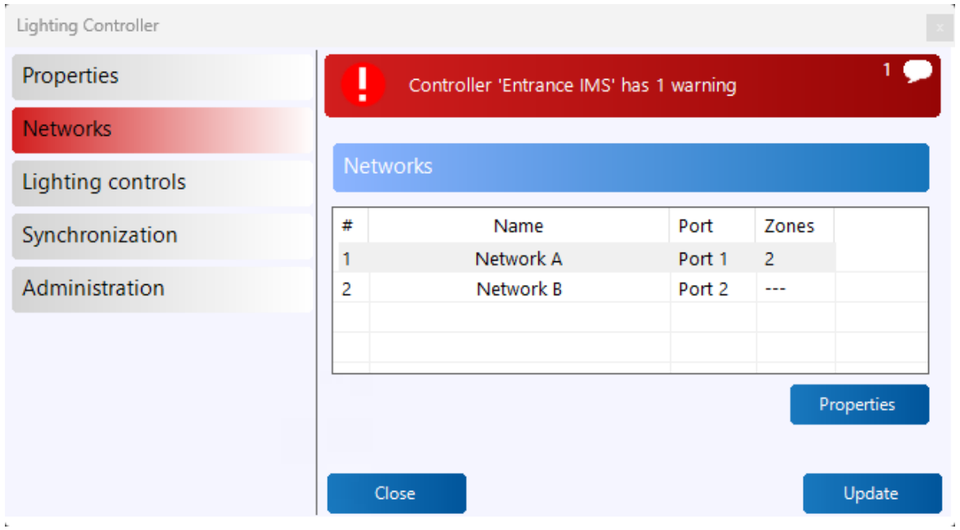


Figure 15.2. *The Networks tab. Network A occupies port 1 and has two zones defined; Network B exists on port 2 but has nothing configured on it yet.*

Four columns carry the whole summary. The number is positional. The name is yours. The port is the physical DALI channel the network is wired to, and mirrors the DALI Channel setting inside the network's own properties — the same fact under two labels. The zones column counts the zones defined on that network, and shows a dash where there are none: a network can perfectly well exist with nothing on it, which is what a freshly created one looks like.

Selecting a row and pressing Properties opens that network. It is worth being clear about what a second network on the same controller actually is, because the tree makes them look like siblings in a way that understates their separateness.

> **TWO NETWORKS ON ONE CONTROLLER SHARE NOTHING**
>
> Each DALI channel is an independent bus. Two networks on the same controller have separate address spaces, separate device budgets — 64 fixtures and 64 input devices *each*, not between them — separate zone numbering, and separate bus power. A device on port 1 cannot be addressed from port 2 by any means.
>
> The only thing they have in common is the controller that drives them, which means they also share its network mode and its daylight harvesting settings. Appendix C §C.3 explains where these limits come from.

15.4 Inside a network

A network's own properties window has four tabs. The first describes the network; the other three — Fixtures, Input Devices and Addressing — are the reconciliation tools covered in Chapter 18.

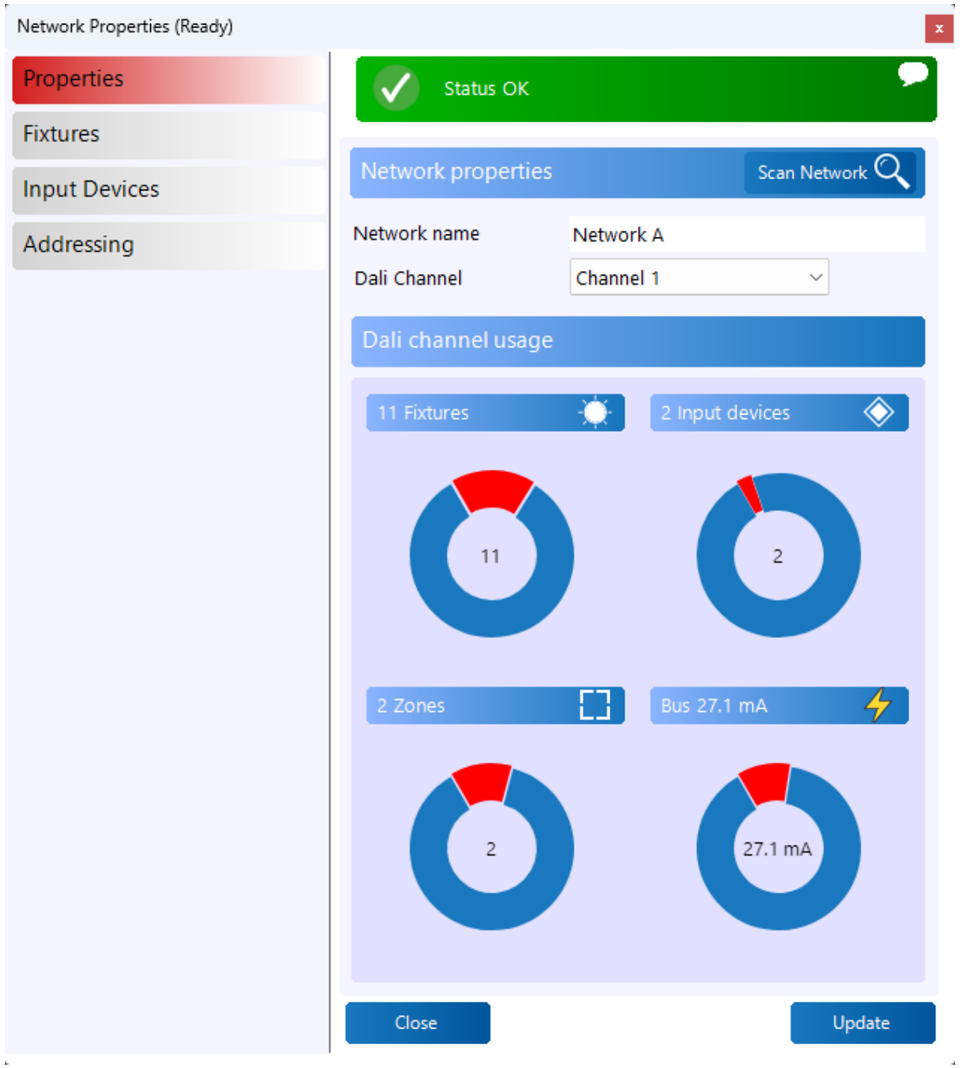


Figure 15.3. *Network Properties. The four dials show how much of each of the network's four separate budgets is in use.*

Two settings and a button sit at the top. The network name is free text. The DALI Channel selects which of the controller's physical channels this network occupies — not a cosmetic setting, since changing it re-points the network at entirely different hardware. Scan Network runs the discovery mechanism described in Chapter 18.

Below them, the DALI channel usage panel shows four dials, and they are more informative than they first appear because each measures a different kind of limit.

Dial	What it is measuring
Fixtures	Control gear in use, against the 64 short addresses available for it.
Input devices	Control devices in use, against their own separate 64 addresses.
Zones	Zones defined, against the 16 DALI groups the bus provides.
Bus	Current drawn from the bus power supply, against what that supply can deliver.

THE FOURTH DIAL IS NOT LIKE THE OTHER THREE

Three of these are counts, and counts are easy to reason about — you either have room for another fixture or you do not. Bus current is electrical, and it is the one that catches people out, because a network can sit comfortably within every count and still be at its limit.

Every device on the bus draws a quiescent current whether or not it is doing anything, and the bus power supply is current-limited — 250 mA is the usual ceiling. Appendix C §C.2 explains why. Watch this dial when adding devices to an established network, particularly one where sensors have been added over time: input devices are small in number but not necessarily small in draw.

15.5 Network mode and the two-tier update policy

Two settings, at two different levels, decide whether configuration held in the MCU is allowed to reach the hardware. Both must permit it. Either one, set restrictively, is enough to stop a write.

- **The building's update mode** — set on the building and applying to everything beneath it. Chapter 13 covers it.
- **The controller's network mode** — set here, and applying to that controller and the networks it carries.

Network mode has three settings.

Mode	Behavior
Disabled	The controller is not contacted at all. No configuration goes down, no status comes up. The controller continues to run whatever it was last given.
Read only	The Gateway connects and reads status, but never writes controller properties. Configuration does not flow down — but direct and quick-test commands still work, so fittings can be exercised to check they respond.
Normal	Configuration flows down to the controller in the ordinary way.

A controller newly added to the MCU defaults to Read only, and this is deliberate rather than cautious housekeeping.

WHY NEW CONTROLLERS ARRIVE READ-ONLY

At the moment a controller is added, the MCU's model of it is empty or half-built, while the controller itself may be carrying a fully commissioned installation that has been running for years. Those two models are about as far apart as they will ever be — and the MCU's is the poorer of the two.

Pushing at that moment would overwrite working configuration with an incomplete one. Read only removes the possibility. It lets you look at what is actually on the bus, reconcile the MCU's model against it (Chapter 18), and satisfy yourself that the two agree — before anything you hold is permitted to reach the hardware.

It is a write barrier on configuration rather than a disconnection: status still flows up, and direct and quick-test commands still work, so you can exercise a fixture to confirm it responds before committing to anything. A controller left in read-only reports a standing warning, which is also deliberate — the mode is a stage to pass through, not a place to leave a commissioned controller sitting.

15.6 Lighting controls

The Lighting controls tab holds the controller-wide settings for daylight harvesting. The behavior itself — what daylight harvesting does, and how it interacts with occupancy and scheduling — is Chapter 21. What lives here are the two numbers that govern how the controller goes about it.

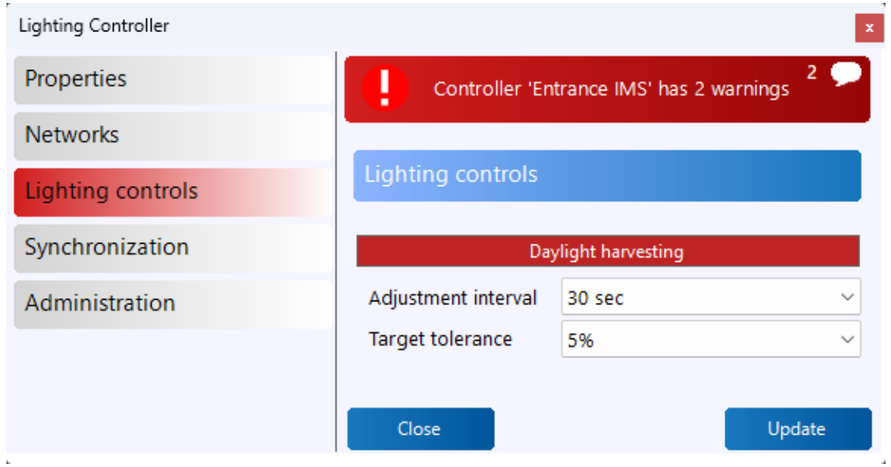


Figure 15.4. *The Lighting controls tab. Both settings apply across the whole controller, not per zone or per network.*

- **Adjustment interval** — how often the controller re-evaluates measured light against the target.
- **Target tolerance** — the dead band. How far measured light may drift from target before the controller does anything about it.

These two settings exist to stop daylight harvesting from hunting, and it is worth understanding the failure they prevent. A daylight-harvesting controller is in a feedback loop with itself: it dims the lights, the sensor measures a room that is now darker partly because of what the controller just did, and it corrects again. Too short an interval, or too tight a tolerance, and the loop never settles — occupants see lights that visibly breathe. Widening the tolerance is usually the better first move, because it addresses the cause rather than merely slowing the symptom. Appendix C §C.7 describes the hysteresis setting that does the same job one level lower, inside the sensor itself.

15.7 Synchronization status

The Synchronization tab reports whether the MCU and the controller currently agree, without changing anything.

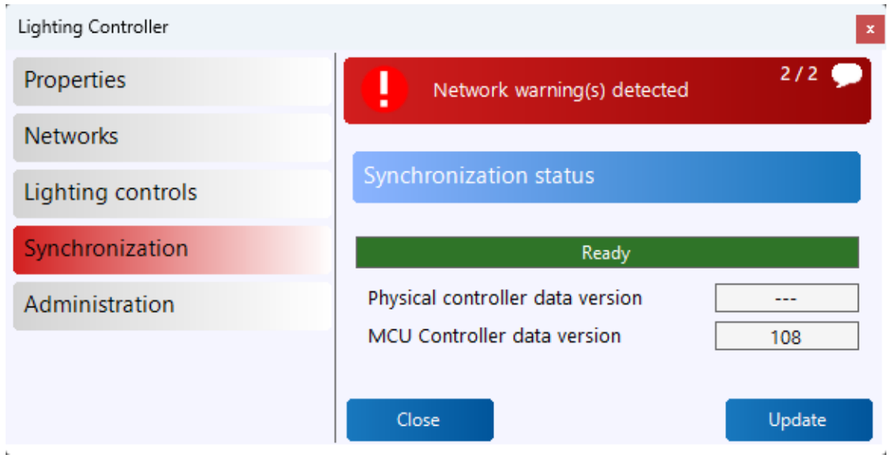


Figure 15.5. *The Synchronization tab. Two revision numbers, one held by each side; a dash means the physical controller's value has not been read.*

The status bar gives the headline. Beneath it sit two data versions: the one the physical controller reports, and the one the MCU holds for that controller. Comparing them is how the system determines whether an update is needed and, if so, in which direction it should flow. A dash in place of the physical value means the MCU has not been able to read it — the controller is unreachable, or its mode prevents the exchange.

Chapter 19 covers what the revision numbers mean, when they advance, and how the comparison decides direction. This tab is the place to look before running any of the reconciliation operations, because it tells you whether there is anything to reconcile.

15.8 Administration and engineering tools

The Administration tab carries the operations that act on the controller.

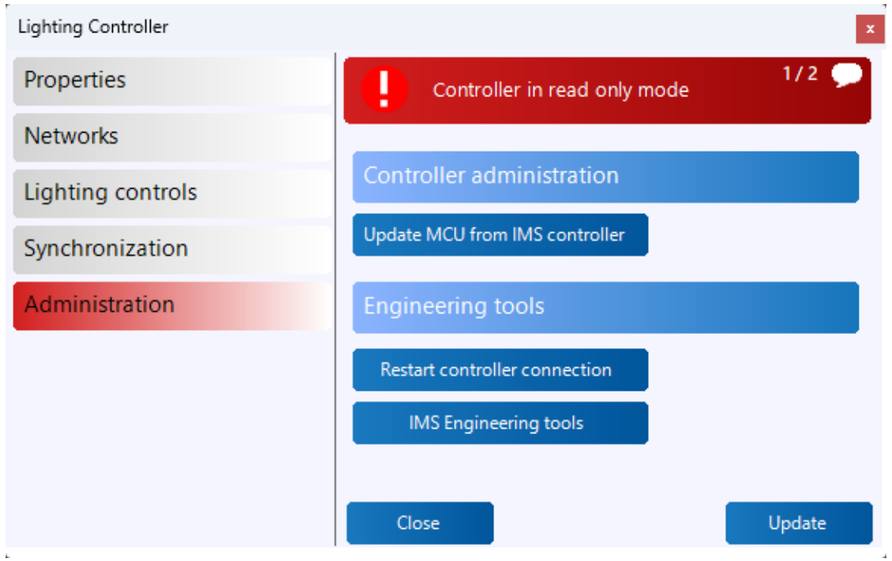


Figure 15.6. *The Administration tab.*

Controller administration

Update controller from IMS pulls the controller's own configuration into the MCU. This is mechanism 3 of the four reconciliation mechanisms, and it has one property worth carrying in your head before you press it: it **prunes but never adds**. Objects the MCU holds that the controller does not know about are removed; objects the controller has that the MCU lacks are not created. Chapter 18 sets out the consequences, including why the MCU's object counts must never be fewer than the controller's before you run it.

Engineering tools

- **Restart controller connection** — tears down and re-establishes the link to the controller. It touches the connection, not the configuration, and is the first thing to try when a controller is present but unresponsive.
- **IMS Engineering tools** — opens the IMS unit's own engineering interface. What lies beyond that point is the controller platform's own tooling rather than the MCU's, and it is not gated by the MCU's network mode.

15.9 The network update menu

Three of the four reconciliation mechanisms are reached not from a properties window but from the area canvas, by right-clicking a network.

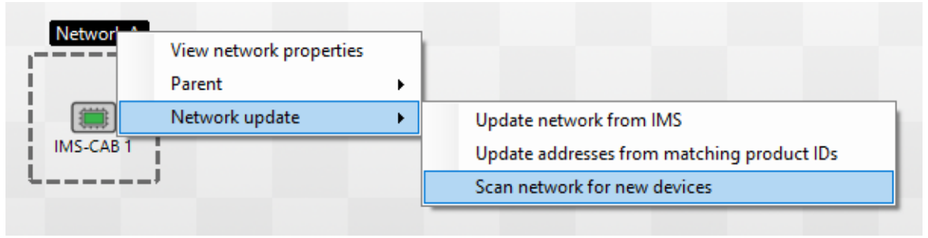


Figure 15.7. *The network context menu, with the Network update submenu open. All three reconciliation mechanisms reachable from the canvas are here. The Network update is only visible if the MCU is connected to the controller.*

- **Update network from IMS** — mechanism 3 at network scope: resolves product IDs, then imports the controller's configuration, pruning without adding.
- **Update addresses from matching product IDs** — mechanism 2. Matches the MCU's records against the devices on the bus by product ID and corrects the short addresses. Requires an exact fixture and input-device count match as a sanity check.
- **Scan network for new devices** — mechanism 4. Discovers what is physically present, keyed on GTIN.

> **THESE ARE NOT INTERCHANGEABLE**
>
> All three appear in one submenu, which makes them look like three routes to the same destination. They are not. Each assumes a different starting position — a network the MCU has never seen, a network whose addresses have shifted, a network that has gained hardware — and running the wrong one can discard work rather than reconcile it.
>
> Chapter 18 sets out which situation each is for, and in what order to reach for them. Do not treat this menu as a troubleshooting sequence to work down.

15.10 Bringing a controller up

Pulling the chapter together, the order below is the one that keeps you out of trouble. Each step is covered above or in the chapter referenced.

1. Add the controller to its area and set its **Name** and **Address**.
2. Leave **Network mode** at **Read only**. Do not change it yet.
3. Confirm the controller is reachable — the status banner clears, and the Synchronization tab reports a physical data version rather than a dash.
4. Add the networks, one per DALI channel actually in use, and set each one's DALI Channel to match the port it is wired to.
5. Reconcile the MCU's model against what is on the bus, using the mechanism appropriate to the situation (Chapter 18).
6. Only once the two sides agree, move Network mode to **Normal**.

> *The controller is where configuration becomes consequence. Everything above it can be edited, discussed and rolled back; below it, lights change in rooms with people in them. Read-only exists so that the transition between the two is a decision rather than a side effect.*

PART III · FOR ENGINEERS

16 Configuring Zones

Chapter 4 explained how zones work. This chapter is its practical counterpart: where each of those ideas is actually configured. A zone's properties window has seven pages, and they are covered here in turn.

16.1 General properties

Open a zone's properties from the floorplan. The first page holds the zone's general settings.

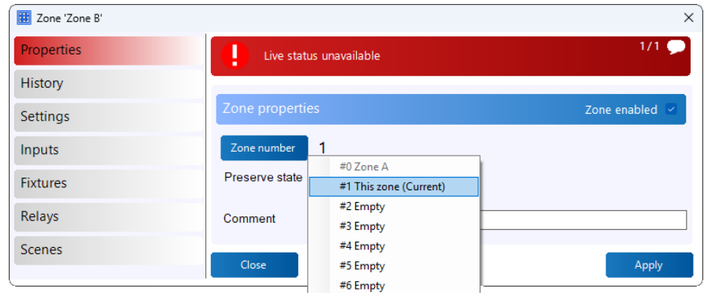


Figure 16.1. *A zone's general properties, with the zone number picker open.*

Setting	Purpose
Zone enabled	When cleared, the controller issues no commands for this zone at all. Its fixtures simply hold whatever level they were last given.
Zone number	The zone's number, 0–15 — and, as Chapter 4 explained, the DALI group its fixtures are bound to. Assigned automatically when the zone is created; it can be reassigned to any free slot, but slots already in use are shown greyed and cannot be taken.
Zone linking	Ties this zone into a synchronized group of zones on other controllers, wired together through an RS485 I/O board. Covered in §16.2. Default: *This zone is not remotely linked*.
Preserve state	When set, the zone resumes its previous state after the controller is power-cycled. When cleared, it starts in Unoccupied.
Comment	Free text for engineering notes.

16.2 Zone linking

Everything else in this chapter keeps a zone to itself — its own sensors, its own fixtures, its own state. **Zone linking** is the exception. It ties a zone into a group of zones spread across **different controllers**, so that when one of them changes state the others follow. A run of stairwells up a building, or a chain of corridors across a site, can then behave as one space even though no single controller sees all of it.

The link is a physical one. Each IMS controller can carry an **RS485 I/O board**, typically with four ports, and the **Zone linking** field chooses which of those ports this zone uses — or leaves it unlinked, the default. The dropdown offers *This zone is not remotely linked* and one entry per port.

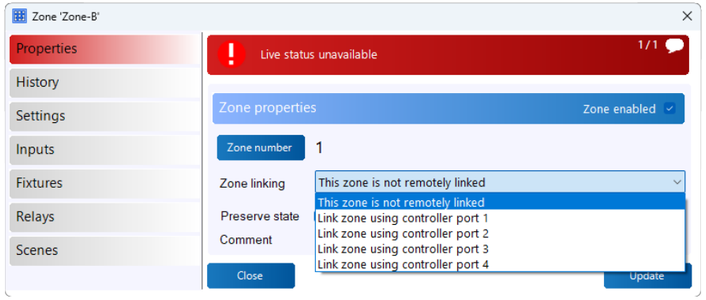


Figure 16.2. *The Zone linking field. Each option names a local I/O-board port; "not remotely linked" leaves the zone independent.*

Once linked, the group shares state continuously. *Any* change in one member — into Occupied, Preliminary, Timeout or Unoccupied — is sent to every other member, which adopts it. If four zones on four separate controllers are linked and one enters Occupied because its motion sensor fires, the other three enter Occupied too. All four states propagate this way, not merely occupied-or-not, so the group stays genuinely in step rather than crudely switched together.

> **THE GROUP IS DEFINED BY THE WIRING, NOT BY THE PORT NUMBER**
>
> This is the part that catches people out. A zone's membership of a group is decided entirely by **which ports are physically wired together**, not by them sharing a number. If port 1 on controller A is wired to port 2 on controller B, then a zone linked to port 1 on A and a zone linked to port 2 on B are in the *same* group — the differing port numbers are irrelevant.
>
> The consequence for anyone reading a configuration: the dropdown tells you which local port a zone uses, but it cannot tell you which group that places it in. That is written on the terminal block, not in the software. Keep the I/O wiring documented alongside the zone configuration, or the grouping becomes impossible to reconstruct.

Linking adds to a zone's behavior; it does not replace it. A linked zone still runs its own occupancy and timeout logic exactly as an unlinked one does — the group is simply another input that can move it. So the mechanism degrades gracefully: if the shared wiring fails, each zone carries on driving its own space from its own sensors, having lost only the coordination between them.

> **NO MCU REQUIRED**
>
> Zone linking lives entirely at the controller level. It is configured on the IMS's own interface and runs over wiring between controllers, so it needs no gateway, no DataManager and no MCU — it is part of the same local autonomy the IMS provides for everything else (Chapter 2). Where the MCU is present and Site Manager is being used, as here, the Zone linking setting synchronizes to and from the controller like any other zone setting (Chapter 19). It is a lateral kind of coordination, controller to controller, quite separate from the central synchronization the MCU performs.

16.3 Where settings come from: zone types

Before configuring a zone directly, it is worth knowing where its values originate. Every zone inherits from a **zone type** — a reusable template defined at building level that describes default behavior. A zone *must* name a zone type, and the system guarantees every building has at least one, so there is always a parent to inherit from.

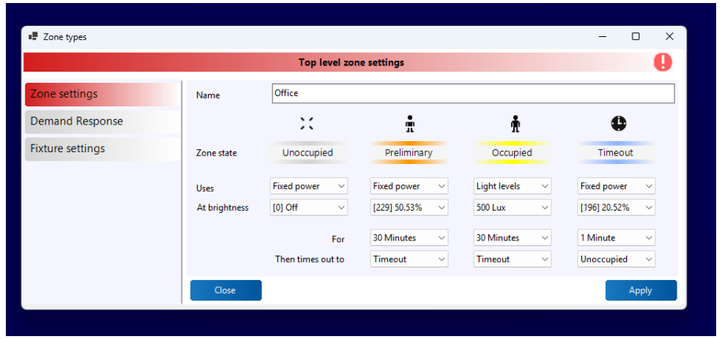


Figure 16.3. *The zone type editor. Its three pages mirror the zone's own settings pages — what is defined here is what zones inherit.*

The pay-off is that common behavior is defined once. Set up an "Office" zone type with the state timings, light levels and demand-response actions your offices should use, point twenty zones at it, and they all behave consistently — while any individual zone can still override a specific value where it genuinely differs.

> **Reading the arrows**
>
> Throughout the settings pages, a value shown in **red with an up-arrow** is inherited from the zone type; a **green left-arrow** beside an editable control means the value is set explicitly on this zone. Clicking the arrow switches between the two, and **Inherit all** returns everything on the page to inherited.

16.4 Zone settings — the state matrix

This is the heart of the zone. The matrix has one column per state, and each column defines what that state does:

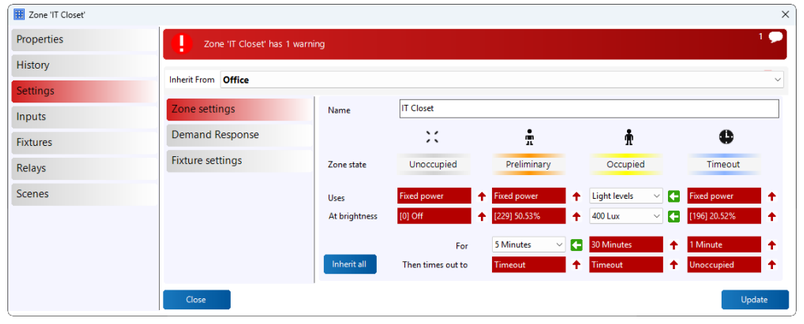


Figure 16.4. *The state matrix: lighting method, level, timeout and destination for each of the four states.*

- **Uses** — the lighting method: fixed power, a light-level target, or a scene.
- **At brightness** — the level to hold. Its meaning follows the method: an arc level for fixed power, a lux /fc target for light levels. For scenes the field disappears, since the scene itself carries the levels.
- **For** — how long the state runs before timing out.
- **Then times out to** — the state it moves to when that timer expires.

Unoccupied has no timeout of its own, because it is the resting floor with nowhere lower to decay to. Every other state may time out to any state you choose — the decay cascade is yours to design, and the usual arrangement is Occupied to Timeout to Unoccupied.

16.5 Demand response and fixture defaults

Two further pages sit alongside the state matrix. **Demand Response** defines what this zone does at each of the three demand-response levels — no action, dim by a percentage, or switch off — applied on top of whatever the zone is currently doing without changing its state.

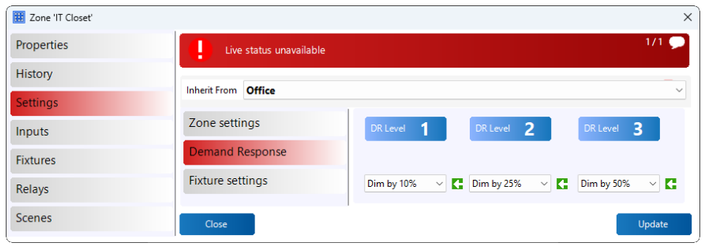


Figure 16.5. *The three demand-response levels and this zone's response to each.*

Fixture settings holds the low-level DALI defaults passed down to the zone's member fixtures — minimum, maximum and system-failure levels, and the fade time and rate. These sit in the middle of the inheritance chain: the zone takes them from its zone type unless set explicitly, and passes them to its fixtures unless a fixture overrides them.

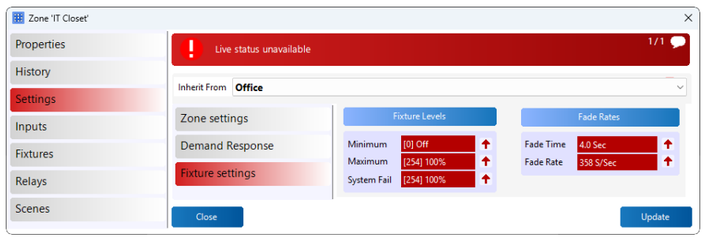


Figure 16.6. *Zone-level fixture defaults. Here every value is inherited from the zone type.*

> **The system failure level matters**
>
> Of these, the system-failure level is the one to think about deliberately: it is what a fixture falls back to if it loses communication on the DALI bus. Whatever the chain resolves to, the driver's own hardware minimum and maximum remain the outer limits — these settings can narrow the range, never widen it.

16.6 Zone Inputs

The Inputs page lists everything that drives this zone, grouped by type — motion sensors, buttons and photosensors — with each contributing instance shown beneath its type and its configured events beneath that.

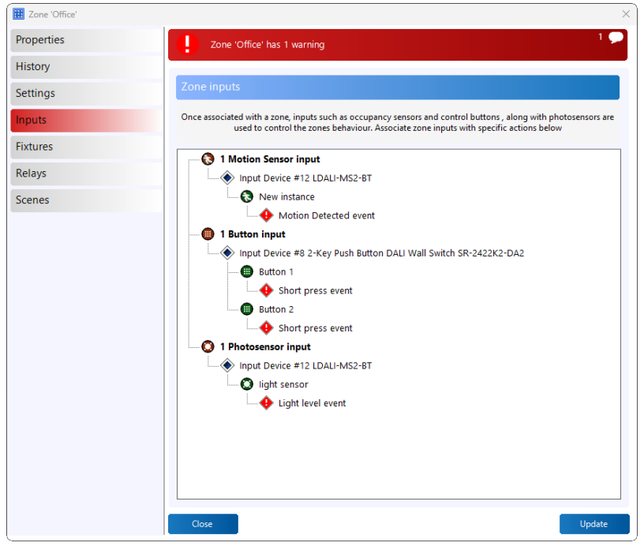


Figure 16.7. *The Inputs tree: each input type, the instances contributing to this zone, and the events configured on them.*

The count against each type is the number of **instances**, not devices — a two-button wall switch contributing both its buttons shows as two button inputs. The instance names come from the descriptions given in the global device catalogue.

Right-clicking an instance adds an event; right-clicking an event opens its **action matrix** — the grid, one column per zone state, that decides what the event does depending on the state the zone is in when it arrives. Chapter 4 explains the thinking behind this at length, including why **No Action** in a particular column is often exactly the right answer. Inputs can equally be linked by dragging them onto the zone on the floorplan, as described in Chapter 14.

16.7 Fixtures and relays

The Fixtures page is a summary rather than an editor. It reports how many fixtures the zone has and how many of them take their settings from the zone rather than overriding with their own, together with a single action that returns every member fixture to inheriting from the zone.

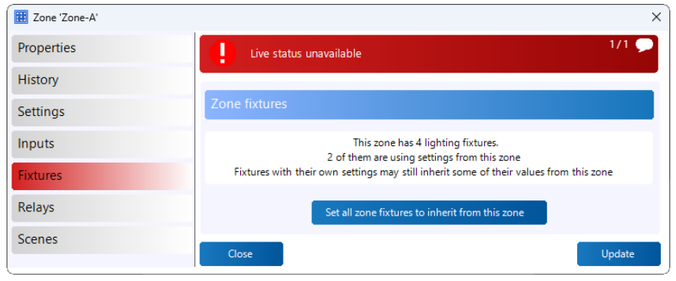


Figure 16.8. *The Fixtures summary, with the action to reset all member fixtures to inherit.*

> **This one applies immediately**
>
> Almost everything in a properties window is staged until you click Update. Resetting the zone's fixtures to inherit is an exception — it prompts for confirmation and then applies straight away, regardless of whether you go on to update the rest of the window.

The Relays page controls any member fixtures whose assigned role is **relay**. These are kept *out* of the zone's DALI group and commanded directly, precisely so their on/off state can be set independently of the lighting level — otherwise any state producing arc power above zero would switch them on.

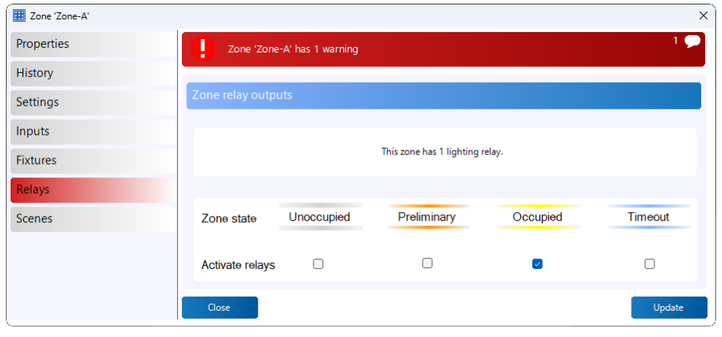


Figure 16.9. *The relay matrix — on or off for each zone state, independent of the lighting.*

The matrix is simple: for each of the four states, are the relays on or off. It applies to *all* of the zone's relays together, and is entirely unaffected by the lighting method or brightness configured for those states.

16.8 Scenes

The Scenes page manages the zone's scene list. Scenes are referred to by index, 0 to 15, and a zone may define up to sixteen. Against each, the page shows how many member fixtures have an explicit level stored for it.

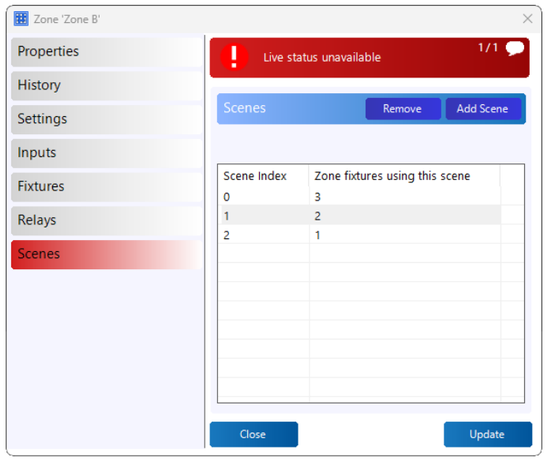


Figure 16.10. *The zone's scene list, with the number of participating fixtures against each index.*

The division of labor is worth being clear about: the *list* lives here at the zone, but the *levels* are stored per fixture. Each member fixture holds its own level for every scene, or "no change" — which is what allows one scene to light part of a zone

while leaving the rest untouched, and why a scene can legitimately exist with no participating fixtures at all. Relay-role fixtures are excluded from scenes entirely.

16.9 History

The last page is diagnostic rather than configuration: a rolling record of what the zone has actually been doing.

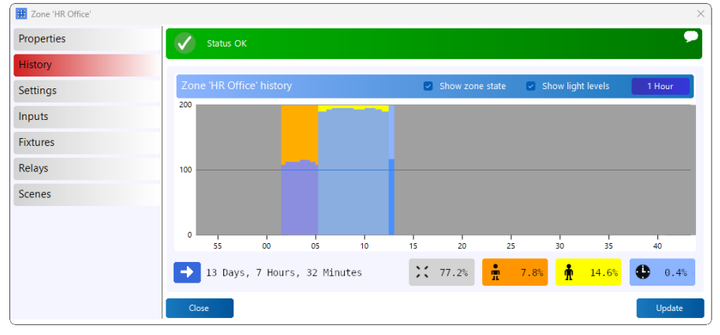


Figure 16.11. *Zone history — state over time, measured light level, and the proportion of time spent in each state.*

The chart plots the zone's state over time, optionally with the light level it measured where a photosensor is present, and the badges below break down what proportion of the period it spent in each state. The view covers either the last hour or the last 24 hours, and the arrow button opens it into its own window — useful for comparing several zones side by side.

This history is held for **24 hours only**, in the gateway rather than the database, and exists purely to help with commissioning and troubleshooting. It is quite separate from the long-term energy data behind the reporting in Chapter 10.

PART III · FOR ENGINEERS

17 Fixtures and Input Devices

Two kinds of object hang off a network, and one distinction between them governs almost everything in this chapter: a fixture is owned by exactly one zone, while an input device is shared by as many as need it. Copy, move and delete all behave differently as a result, and so does the way each is configured. This chapter covers both objects tab by tab, and the canvas operations that apply to them.

17.1 Owned, and shared

A fixture is control gear: something that makes or switches light. An input device is a control device: something that senses or requests. Appendix C §C.4 explains why DALI draws that line so firmly. The MCU inherits it, and adds one of its own.

- **A fixture belongs to exactly one zone.** Its zone is what controls it, and it cannot be controlled by two. Zone membership is a DALI group membership held on the device itself (Appendix C §C.3).
- **An input device is shared.** A sensor in a corridor may inform several zones at once, and a wall plate may have one button acting on one zone and another acting on a different one. Nothing owns it.

> **THIS IS WHY COPY, MOVE AND DELETE DIFFER**
>
> Because a fixture has exactly one owner, moving it is unambiguous — it leaves one zone and joins another — and deleting it removes something only one zone was relying on.
>
> An input device has no single owner, so the same operations have to be thought about differently: deleting one may affect several zones at once, and none of them will necessarily be the zone you had open at the time. Where the two objects behave differently in this chapter, this distinction is almost always the reason.

17.2 A fixture's properties

Opening a fixture gives four tabs: Properties, Fixture settings, Scenes and Addressing. The first describes what the fixture is.

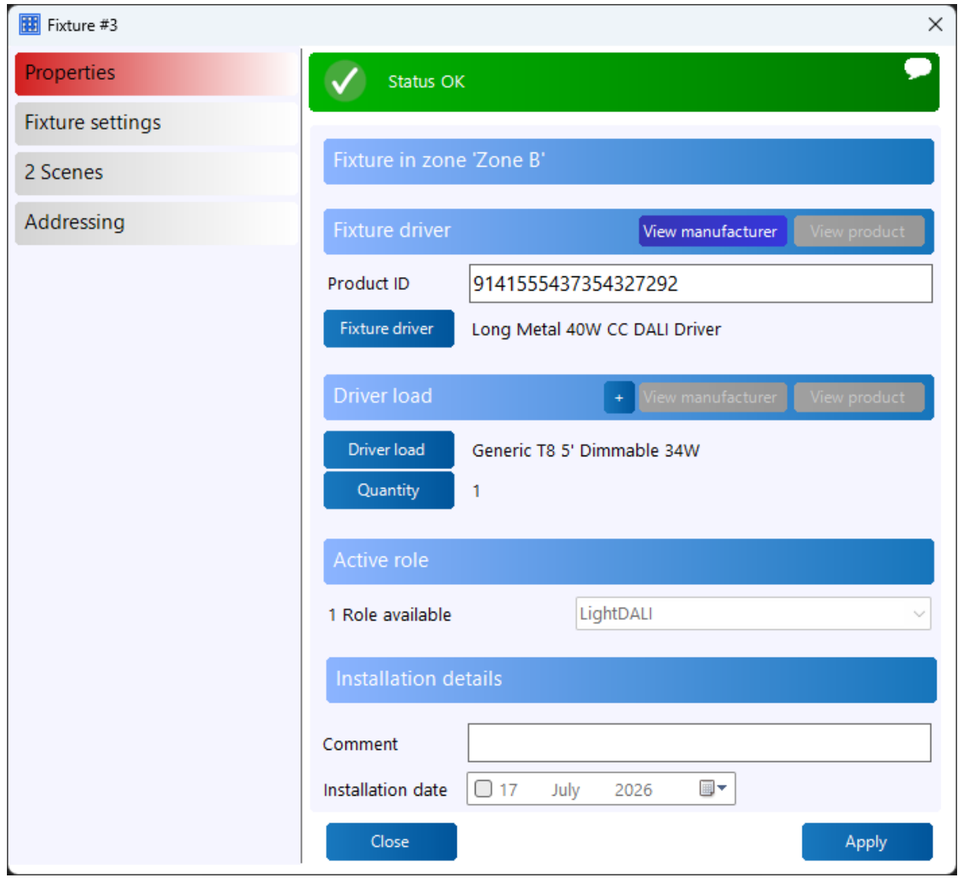


Figure 17.1. *A fixture's Properties tab. The banner at the top names the zone that owns it. This capture predates two renames: the role field is now Assigned role, and the commit button now reads Update.*

- **Fixture driver** — the control gear itself, drawn from the device catalogue (Chapter 27). The View manufacturer and View product buttons open the catalogue entry behind it, which is the quickest way to check what a device is actually capable of.
- **Driver load** — what the driver is driving: the lamp type, and how many of them. This is not cosmetic. Energy figures are estimated from load-type power curves rather than metered (Chapter 10), so the load recorded here is what the reported consumption for this fixture is calculated from. A driver left with the wrong load reports plausible numbers that are wrong.
- **Assigned role** — how the fixture is driven: DALI, 0-10V or Relay. Appendix C §C.10 maps each onto the class of control gear behind it. The count beside it reports how many roles this device supports; where a device offers only one there is nothing to choose.
- **Product ID** — the identifier that distinguishes this individual unit from others of the same model, and what reconciliation matches on (Chapter 18).
- **Installation details** — a comment and an optional installation date. Worth filling in: it is the only place a fixture's age is recorded, and lamp and driver lifetimes are the usual reason for wanting to know it.

17.3 Fixture settings, and when to use them

The second tab holds the DALI values that govern how this fixture behaves — and, more importantly, whether it uses its own values at all.

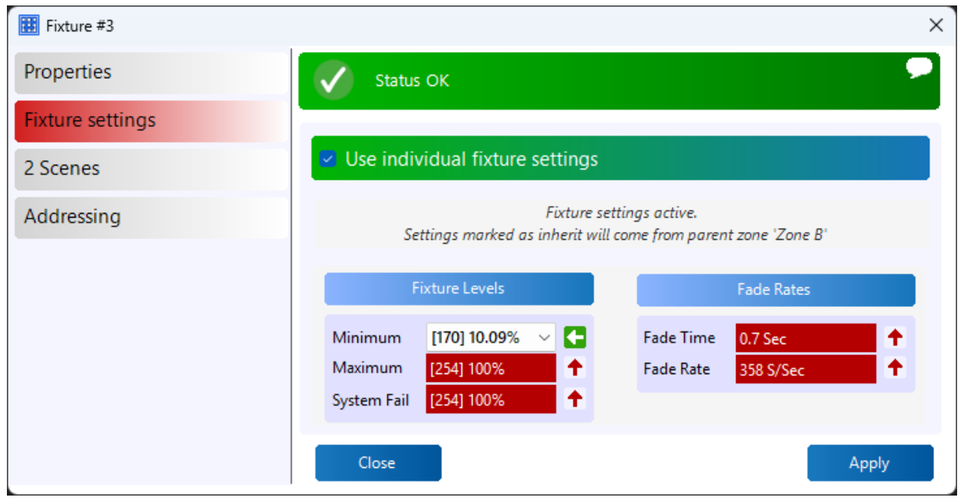


Figure 17.2. *Fixture settings. The checkbox at the top is the master switch; the arrows beside each value say where it is coming from.*

Use individual fixture settings is the switch. Clear it and the fixture takes everything from its zone. Set it, and the panel becomes live — but note what the caption underneath says: settings still marked as inherited continue to come from the parent zone. Turning the switch on does not detach every value, it merely allows individual values to be detached.

Each value carries the inheritance markers described in Chapter 5 §5.2 — a red up-arrow for inherited, a green left-arrow for explicit. In the figure only Minimum has been set on the fixture itself; everything else is still coming from zone 'Zone B'.

Setting	What it does
Minimum	The lowest level this fixture will go to. Commanded below it, the driver settles here rather than switching off — only an explicit zero turns it off.
Maximum	The highest level it will go to.
System Fail	Where the fixture goes if the bus itself fails. Deliberately not "off", so a cut cable leaves people with light.
Fade Time	How long the fixture takes to reach a commanded level.
Fade Rate	How fast it moves when told to fade up or down without a stated destination — the behavior behind a held dimming button.

> **TWO NUMBERS, ONE LEVEL**
>
> Levels are shown as a raw DALI value in brackets followed by a percentage — `[170] 10.09%`. The percentage is actual light output; the bracketed number is the position on the logarithmic curve that produces it. They deliberately do not track one another, and Appendix C §C.5 explains why. Read the bracketed value as a position on a curve, never as a percentage.
>
> System Fail is worth a second look while you are here. It is the one setting on this tab that matters most when the control system is *not* working — the fixture holds it and acts on it alone, with no controller involved.

17.4 Fixture scenes

A scene is a level stored inside the device against a scene number, recalled by a single command so every fixture in a zone moves together. Appendix C §C.6 covers the mechanism; this tab is where a fixture's own contribution to each scene is set.

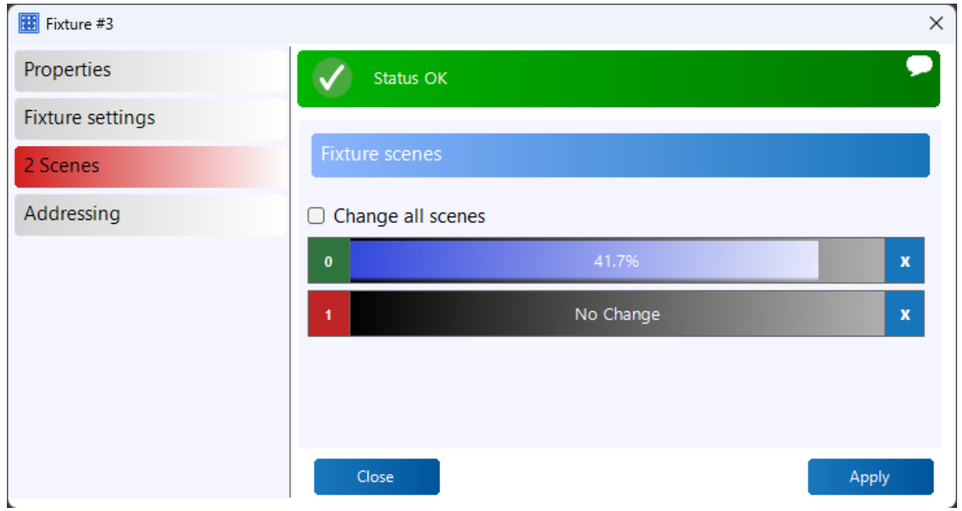


Figure 17.3. *A fixture's scenes. Scene 0 holds a level; scene 1 is set to No Change, meaning this fixture does not participate.*

Each scene the zone defines appears as a row with a slider. Change all scenes applies an adjustment across every row at once, which is quicker than working through them individually when a fixture simply needs to sit lower than its neighbors throughout.

“NO CHANGE” IS A SETTING, NOT A GAP

A row reading No Change is the DALI MASK value. Recalling that scene leaves this fixture exactly where it was — it is a deliberate instruction to stay put, not an unconfigured row waiting to be filled in. This is how a scene can address part of a zone and leave the rest alone.

It follows that scene levels live in sixty-four separate devices rather than centrally. A fixture that has been replaced arrives from its box knowing nothing about the scenes it is expected to join, which is a routine cause of one luminaire behaving oddly after a swap.

17.5 Fixture addressing

The fourth tab shows the fixture address space for the network, with this fixture’s current address outlined and the addresses in use elsewhere shown so a conflict cannot be created by accident. Any address with a problem will be highlighted in red, hover over an address to show a popup with address details.

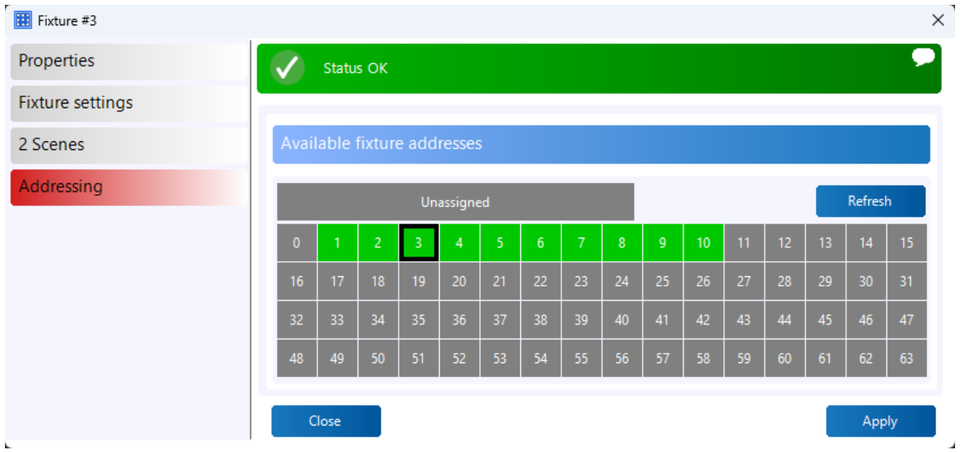


Figure 17.4. *A fixture’s Addressing tab. Changing an address here affects one device; Chapter 18 covers the network-scope operations.*

Use this where a single device is wrong and the rest of the network is fine. Where the discrepancy is wider than one device, the network-scope mechanisms in Chapter 18 are both faster and safer than working through them one at a time.

17.6 An input device’s details

Input devices open on a Details tab rather than a Properties one, because the first thing you usually need to know about a control device is not what it is but how much of it is being used.

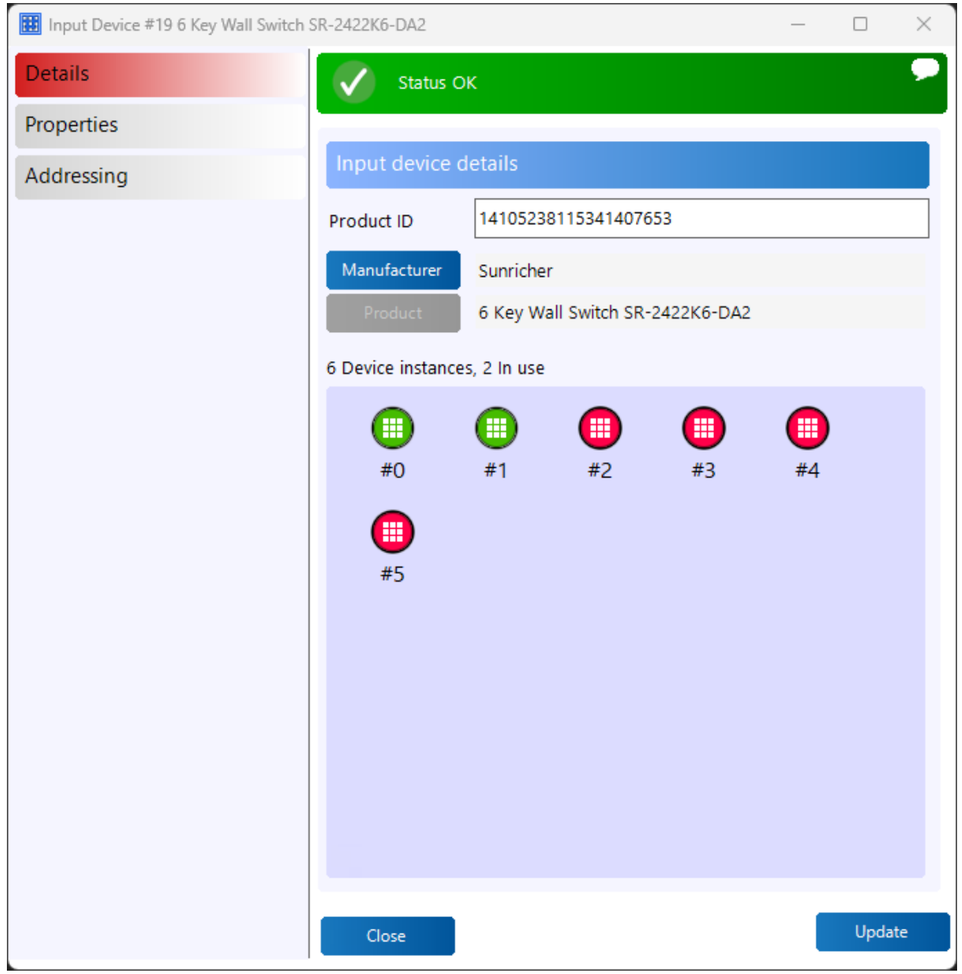


Figure 17.5. *An input device’s Details tab. Six instances exist on this six-key wall switch; two are in use.*

Product ID, manufacturer and product come from the catalogue as they do for a fixture. Below them sits the instance grid. An instance is one sensing or input element within the device — one button, one occupancy sensor, one light sensor — and a device commonly presents several (Appendix C §C.7). The count states how many exist and how many are in use, and the grid shows which: those doing a job are marked apart from those that are not.

ONE DEVICE, SEVERAL JOBS, NO OWNER

This screen is the clearest illustration of why input devices are shared rather than owned. A six-key wall switch is six independent inputs in one enclosure, and there is no reason all six should act on the same zone. A multi-sensor is the same story with an occupancy instance and a light-sensor instance.

So the question "which zone does this device belong to?" has no answer. The useful question is which instances are in use, and that is what this tab reports.

17.7 Instance settings

Each instance carries its own settings, reached from the instance itself rather than from the device as a whole.

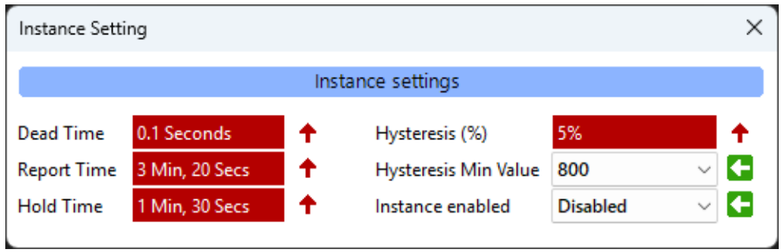


Figure 17.6. *Instance settings. The timers are the DALI-2 sensor timers; the inheritance arrows work exactly as they do everywhere else.*

- **Dead Time** — a short interval after a report during which further reports are suppressed, so a busy corridor does not saturate the bus.
- **Hold Time** — how long the instance itself continues to treat the space as occupied after the last movement.
- **Report Time** — how often that continued occupancy is re-reported.
- **Hysteresis** and **Hysteresis Min Value** — for light-sensor instances, how far measured light must move before the instance reports a change. This is what stops a sensor oscillating around its target as the fixtures it controls change the light it is measuring.
- **Instance enabled** — whether this instance is active at all.

HOLD TIME WILL OVERRIDE ANY ZONE TIMEOUT YOU SET

These timers sit *below* the zone state machine and are independent of it. If a sensor's own Hold Time is longer than the zone's timeout, the zone never sees the silence it is waiting for, and the timeout achieves nothing — the zone stays occupied because the sensor is still reporting.

When a zone refuses to time out at the interval configured for it, this is the first place to look, not the zone settings. Appendix C §C.7 explains why occupancy sensors behave this way at all.

Instance enabled deserves particular note, because it is how a single broken input is taken out of service without touching anything else. A button that has failed can be disabled where it sits, leaving the other five on the same plate working and the device itself in place and addressed.

17.8 Properties and addressing

The remaining two tabs are short. Properties holds a comment; Addressing works exactly as it does for a fixture, against the separate input device address space.

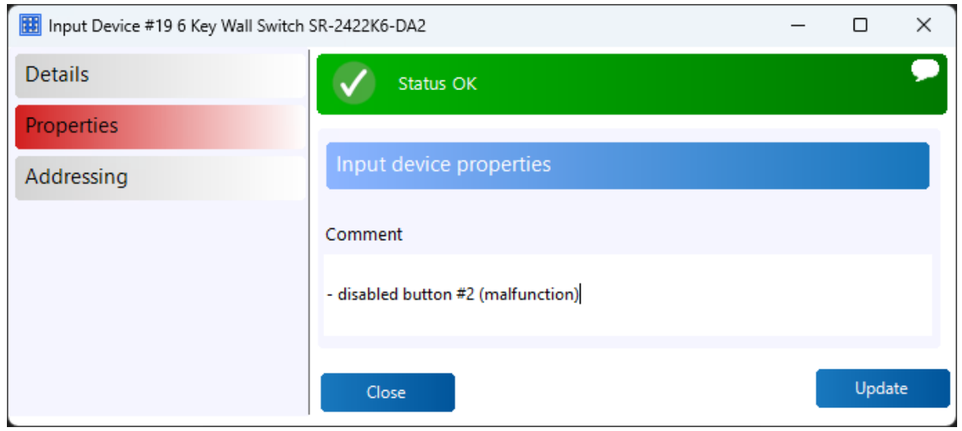


Figure 17.7. *The Properties tab. Pairing a disabled instance with a comment recording why is a habit worth forming.*

The comment in the figure is doing real work. An instance disabled without explanation looks, six months later, like a configuration error somebody should correct — and correcting it puts a known-faulty button back into service. A line of text prevents that.

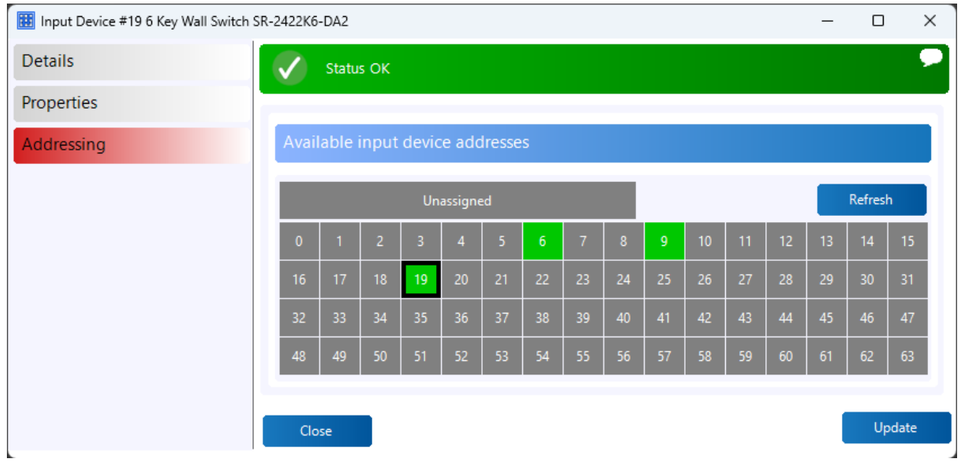


Figure 17.8. *An input device's Addressing tab, showing the input device address space.*

The two address spaces are separate, as Appendix C §C.3 explains, so a fixture and an input device may both sit at address 6 on the same network without conflict. This surprises people who expect a single pool of sixty-four.

17.9 Working on the canvas

Both objects carry a right-click menu on the area canvas, and it is where the structural operations live.

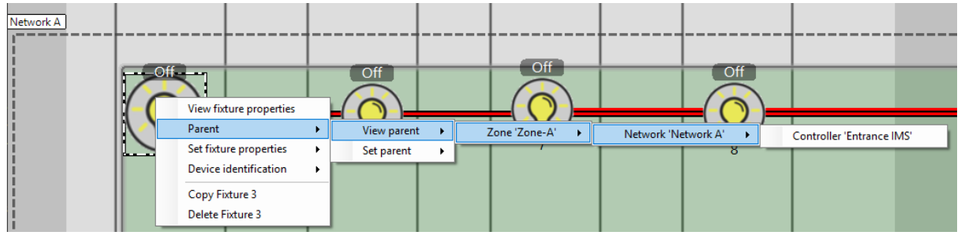


Figure 17.9. *A fixture's context menu with the Parent submenu open, walking up from the fixture through its zone and network to the controller.*

- **View parent** — walks up the hierarchy, offering the zone, then the network, then the controller. Useful for orientation on a large plan when you have found a fixture and need to know what it hangs off.
- **Set parent** — moves the fixture to a different zone. See the callout below.
- **Set fixture properties** and **Device identification** — property editing and physical identification without opening the full properties window.
- **Copy** and **Delete** — named for the specific object, so there is no ambiguity about which one is about to be affected.

> **CHANGING A FIXTURE'S ZONE IS A CHANGE TO THE HARDWARE**
>
> Set parent looks like a filing operation — moving an object from one part of a tree to another. It is not. A zone is a DALI group, and a fixture's zone membership is held *on the device itself* (Appendix C §C.3). Moving a fixture between zones rewrites group membership on the driver in the ceiling.
>
> The practical consequences follow from that. The move has to reach the hardware to take effect, so it will not do anything on a controller in Read only mode; and the fixture's scene levels, which are also held in the device, do not follow it — the new zone's scenes are the ones it now needs to hold.

For a **fixture**, Set parent offers two quite different kinds of move, and it is worth knowing which you are making. Re-zoning — choosing **No zone** or another zone **on the same network** — is the everyday operation, the logical reorganization of a building as spaces change use. Re-networking — choosing **another network on the same controller** — is the rarer one, for when a device has been physically rewired onto a different bus.

> **CHANGING A FIXTURE'S NETWORK UNZONES IT**
>
> The two moves are not symmetrical. Re-zoning keeps the fixture on its network and simply changes which zone drives it. Moving a fixture to a different **network**, though, automatically leaves it **unzoned** — the zones it belonged to were on the old network and cannot follow it across. After a re-network, the fixture has to be given a zone on its new network before it does anything.
>
> Neither move can cross to a different controller. A fixture is a device on one controller's DALI bus, so the menu offers only destinations that device could actually reach — the same boundary the wiring imposes.

For an **input device**, Set parent is more restricted, and the restriction follows from the fact that an input device is shared rather than owned (§17.1). It can be moved to another network on the same controller exactly as a fixture can — but only while **no zone is using it**. While any zone still listens to it, the option is unavailable.

> **AN IN-USE INPUT DEVICE CANNOT BE MOVED**
>
> A fixture belongs to one zone, so reassigning it is unambiguous. An input device may serve several, and moving it would silently break every one of those links at once — so the software will not let you. To re-network an input device that is in use, detach it from its zones first (from each zone's Inputs page, or by deleting the links on the canvas), then move it.

Zones have no Set parent at all. A zone *is* a DALI group on a particular network (Appendix C §C.3); there is no sense in which it could be moved to a different one while remaining the same zone. Where a zone needs to exist elsewhere, it is created there.

17.10 Triggering an input by hand

An input device can be made to raise its events from the canvas, without anyone walking to the switch or waving at the sensor.

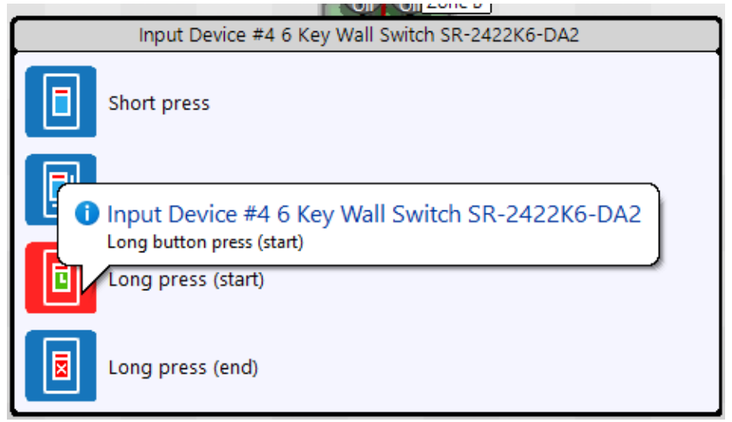


Figure 17.10. *The event-trigger popup for a six-key wall switch, offering the button events the instance can raise.*

The events offered are those the instance supports — for a button, a short press, a double press, and the start and end of a long press. Raising one drives the zone exactly as the physical input would, through the state and action matrix described in Chapter 16.

> **THE FASTEST WAY TO TEST AN ACTION MATRIX**
>
> The matrix in Chapter 16 decides what an event does according to the zone's current state, which means proving it behaves correctly requires putting the zone into each state in turn and firing the event from each one. Doing that physically means a great deal of walking and waiting for timeouts.
>
> Triggering events from here collapses that into a few clicks at the screen, with the zone's state visible on the plan at the same time. It is worth doing on any matrix more complicated than "any button turns it on".

> *A fixture is owned and an input device is shared. Almost every difference in how the two are configured, copied, moved and deleted comes back to that one sentence.*

PART III · FOR ENGINEERS

18 Device Reconciliation and Addressing

The MCU holds a model of what is on each DALI bus. The bus holds the devices themselves. These two models drift apart, and everything in this chapter exists to bring them back together. The work is not difficult, but it is unforgiving of being done in the wrong order — so this chapter is organised around deciding which mechanism a situation calls for, rather than around the buttons themselves.

18.1 What reconciliation is for

A network in the MCU is a model: a set of fixture and input device records, each carrying a short address saying where on the bus that device is expected to be. The bus itself carries the actual devices. Nothing keeps these two in step automatically, and there are more ways for them to diverge than most people expect.

- The installation was commissioned before the MCU knew the site existed, so the bus is fully addressed and the model is empty.
- A driver failed and was replaced. The new one arrived with no address, or with the wrong one.
- Somebody re-addressed the bus, and every address moved at once.
- Hardware was added — a sensor in a meeting room, a run of fittings in a new partition — and the MCU has never seen it.
- The controller was configured directly through its own tooling, and the MCU was not told.

Appendix C §C.8 sets out the two identifiers that make reconciliation possible, and the distinction is worth restating because every mechanism in this chapter turns on it. The **GTIN** identifies the model — what kind of thing this is — and is fixed in the device's firmware. The **Product ID** identifies the individual unit — which particular one this is — and exists both in the device and against the MCU's record of it. The **short address** identifies only *where* the device currently sits, and it is the thing free to move.

> **THE PRINCIPLE UNDERNEATH ALL FOUR MECHANISMS**
>
> Reconciliation never starts from addresses. It establishes *identity* first and corrects the addresses afterwards. An address is the least durable fact about a device; treating it as the thing to trust is what turns a small discrepancy into a network that has to be commissioned again from scratch.

18.2 Reading the addressing grid

The Fixtures and Input Devices tabs of a network's properties each show the same thing for their own kind of device: sixty-four cells, one per short address, describing where the MCU's model and the bus agree and where they do not.

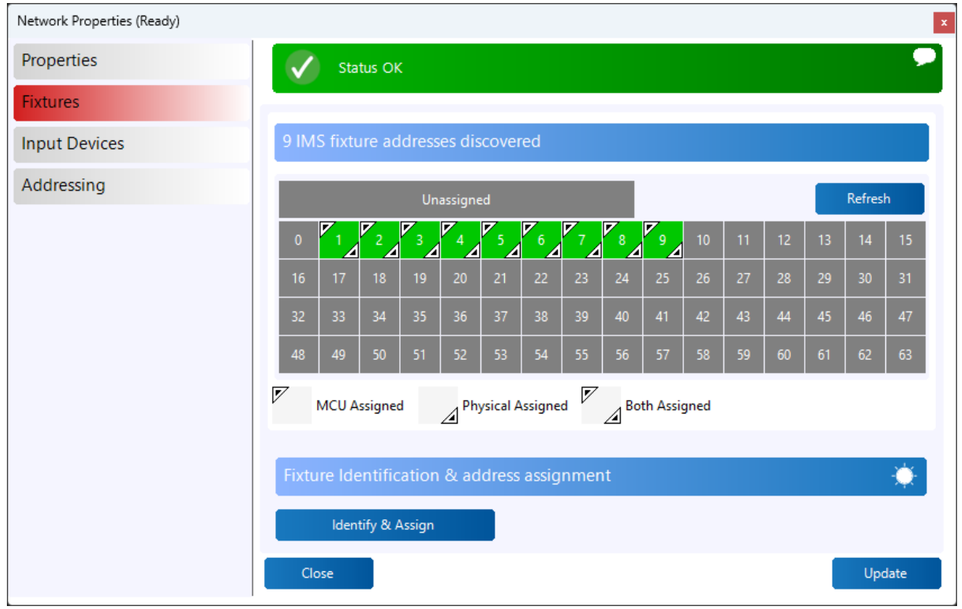


Figure 18.1. *The Fixtures tab. The banner reports how many addresses were found on the controller side; the grid shows how those compare with what the MCU holds.*

The legend beneath the grid names the cell states, and the distinction between them is the whole value of the screen. The first three describe how the two sides compare at an address; the fourth flags a clash that has to be cleared up.

Cell state	What it tells you
MCU Assigned	The MCU believes a device sits at this address. The bus does not agree. Something was removed, has failed, or has been re-addressed.
Physical Assigned	A device is present at this address that the MCU knows nothing about. Something was added, or was never imported.
Both Assigned	The two sides agree. Nothing to do.
Duplicate address	Two or more devices share this address. Flagged with a **!!** icon, and — like any warning — the cell is shown in red. The duplication may be on the MCU side, on the physical bus, or both; hovering over the cell says which. Needs resolving: see §18.3.

The bar above the grid accounts separately for devices that have no short address assigned at all. A device there is present on the bus and detectable, but has no place in the addressing scheme yet — the ordinary state of hardware straight out of its box, and what Re-Address NEW (§18.10) exists to resolve. Until it has an address it cannot appear in the grid, because the grid is organised by address.

Refresh re-reads the bus. It is worth pressing after any operation that changes addresses, because the grid is a snapshot rather than a live view.

THE GRID IS A DIFF, NOT AN INVENTORY

It is tempting to read this screen as a list of what is on the network. It is not — it is a comparison, and its entire value lies in the cells where the two sides disagree. A grid showing sixty-four cells in agreement tells you almost nothing except that there is no work to do.

Read it by asking which side each lonely cell is on. Cells the MCU claims alone describe things that have gone; cells the bus claims alone describe things that have arrived. That single question usually identifies what happened without any further investigation.

18.3 Duplicate addresses

Every address in the grid is meant to belong to at most one device. When more than one lays claim to it, the grid marks the cell with a **!!** icon and colours it red — as it does for any warning. A duplicate is not a difference between the two sides in the way the other cell states are; it is a genuine fault, and it will block clean addressing until it is cleared.

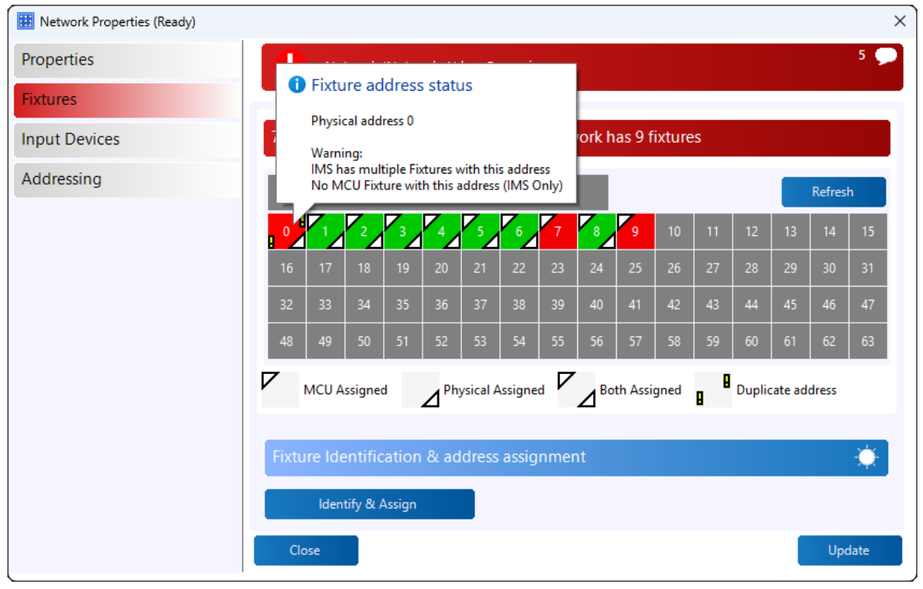


Figure 18.2. *A duplicate at address 0. The cell carries the !! icon and the hover panel names the problem — here, several IMS fixtures share the address while no MCU fixture holds it.*

Hovering over a flagged cell shows its status, and the wording matters, because a duplicate can arise on either side — or both — and the fix differs.

On the MCU — two or more of the MCU's own device records have been given the same short address. Nothing on the bus need be wrong; the model is simply inconsistent with itself.

On the physical bus — two or more real devices are responding at the same address. This is the more serious case: the bus itself cannot address them apart, and they must be physically re-addressed before anything else can be trusted.

Right-clicking a duplicated cell opens a context menu whose options depend on which kind of duplicate it is. Two actions are possible.

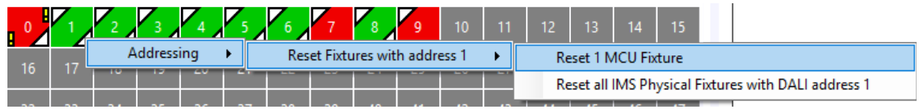


Figure 18.3. *The right-click Addressing submenu on a duplicated address. The two actions name the address and, for the MCU reset, the number of records affected; the physical reset appears only with the IMS connected.*

Reset the MCU fixtures — shown as *Reset N MCU Fixture*, giving the count. It removes the clashing short address from every MCU record using it, leaving those devices unassigned. Use it for an MCU-side clash; the devices can then be given distinct addresses again from their own Addressing tabs (§18.11).

Reset the physical devices — shown as *Reset all IMS Physical Fixtures with DALI address N*, and offered only when the IMS is connected. It clears the physical short address from every device answering at that address, leaving them unassigned on the bus. They are then re-addressed as new devices from Controller Properties →Addressing (Re-Address NEW / unassigned devices, §18.10).

MATCH THE FIX TO THE SIDE THE DUPLICATE IS ON

The hover panel is telling you which action you need. A duplicate that exists only on the MCU is a records problem — **Clear the MCU addresses** and reassign. A duplicate on the physical bus is a hardware problem — **Physically reset** the devices and re-address them. A duplicate that shows on both needs both, physical first: there is no point reassigning MCU records to match a bus that still cannot tell its own devices apart.

Physically resetting needs the IMS connected, for the obvious reason that it reaches out and rewrites the devices. If the option is greyed, the controller is unreachable and only the MCU-side clear is available.

18.4 When the counts do not match

Alongside the per-address comparison, the banner reports a count — and where the two sides hold different numbers of devices, it says so explicitly rather than leaving you to total the cells.

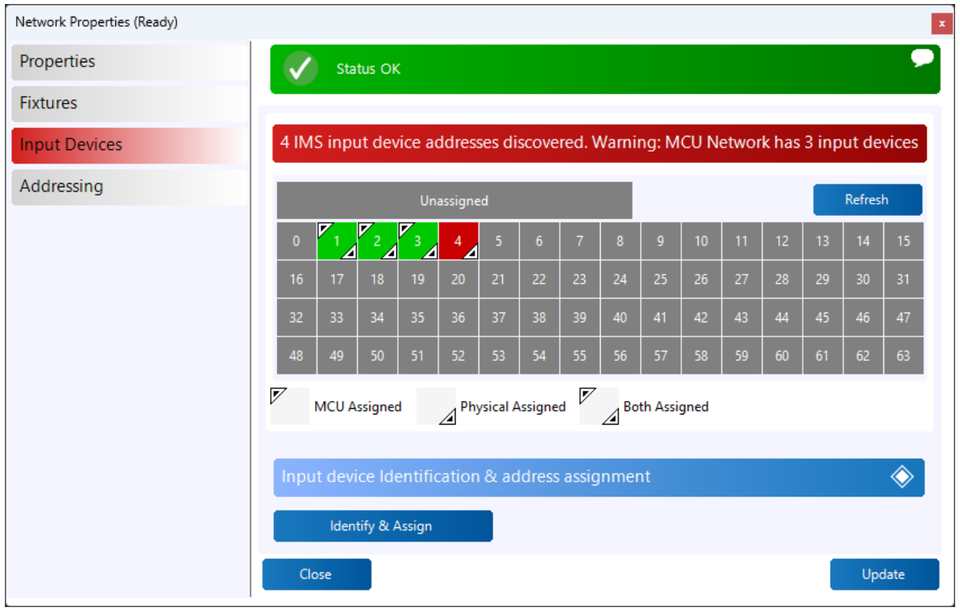


Figure 18.4. *The Input Devices tab reporting a count mismatch: four devices found on the controller, three held by the MCU. The address in red is the one with no counterpart.*

This is called out separately from the cell states because it is not merely a symptom. It is a precondition, and it decides whether one of the four mechanisms may be used at all.

> **WHY AN EXACT COUNT MATCH IS DEMANDED**
>
> Matching by Product ID is a pairing operation: every record on one side is matched to exactly one device on the other. If the two sets are different sizes, at least one member has no counterpart — and a pairing that proceeds anyway must either leave something unmatched or match it to the wrong partner.
>
> The count check is a cheap test that catches this before any address is rewritten. It applies to fixtures and input devices *separately*: both must match exactly. If the banner is reporting a mismatch, the population is wrong, and the population has to be fixed before addresses are worth touching.

18.5 The four mechanisms

There are four ways to bring the two models together. They are not four routes to the same place: each assumes a different starting position, and each trusts a different thing.

	Mechanism	What it trusts	Where it is reached
1	Identify & Assign	Your eyes	Fixtures / Input Devices tab
2	Update addresses from matching product IDs	Product ID — the unit	Network context menu
3	Update network / controller from IMS	The controller's own configuration	Context menu; controller Administration tab
4	Scan network for new devices	GTIN — the model	Scan Network; context menu

18.6 Mechanism 1 — Identify & Assign

When there is nothing reliable to match on, the remaining option is to look. Identify & Assign makes one device on the bus announce itself physically, and asks you to say which record in the MCU it corresponds to.

The controller steps through the physical devices the IMS reports, identifying them one at a time. A fixture being identified cycles on and off once per second; an input device typically blinks an LED. While a device is signalling, you click the matching object on the floorplan and confirm, the MCU records that object's short address to match, and the controller advances to the next device. Where the physical device count differs from the MCU's configured count, a confirmation gate appears once at the very start, before identification begins.

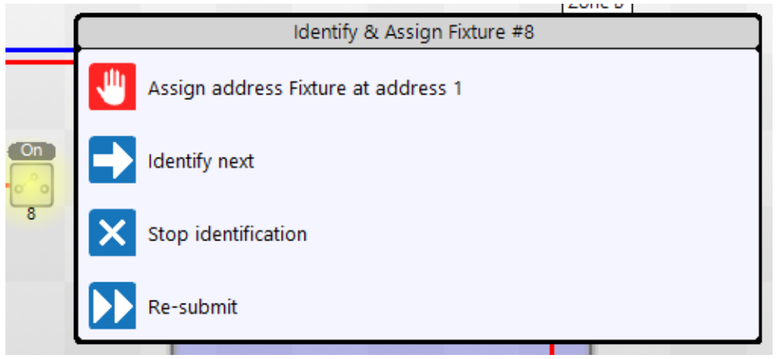


Figure 18.5. *The Identify & Assign popup, shown on the floorplan beside the device currently being identified.*

- **Assign address** — binds the MCU object named in the title to the physical address currently identifying itself. This is the decision the whole mechanism exists to capture.
- **Identify next** — skips the current device and moves on to the next one.
- **Stop identification** — ends the run.
- **Re-submit** — re-queues a device that was assigned in error, putting it back into the run so the assignment can be made again correctly.

SLOW, AND AUTHORITATIVE

This is the only mechanism that observes rather than infers. The other three reason from data that may itself be wrong — a Product ID that was never written, a controller configuration that is stale, a GTIN shared by a device that was swapped for a different one.

That makes Identify & Assign the tie-breaker. When the automatic mechanisms disagree with one another, or when a network is in a state nobody can account for, this is what settles it — it is the one answer that cannot be wrong about which fitting is which.

It is laborious, and on a large network it has traditionally meant two people: one at the screen, one standing under the fittings. Running Site Manager on a tablet over the site wireless collapses that back to one, which is worth arranging before starting on anything larger than a few dozen fittings.

18.7 Mechanism 2 — Update addresses from matching product IDs

Because the Product ID sits on both sides — in the device, and against the MCU's record of that device — it can be matched across the gap. This mechanism pairs the MCU's records with the devices found on the bus by Product ID, then rewrites the MCU's short addresses so they say where those devices *actually* are.

It is the right tool for exactly one situation: the addresses have moved, but the population has not. After a re-address, after a controller has been swapped, after any operation that shuffled the bus without changing what is on it, this restores the mapping in a single step.

It requires an exact fixture count match and an exact input device count match, for the reason given in §18.4. It is therefore the wrong tool immediately after hardware has been added or removed — correct the population first, then use this to fix the addresses.

NOTHING ON THE BUS CHANGES

This mechanism edits the MCU's model and nothing else. No device is re-addressed, no configuration is written, and no physical hardware is touched — it changes only what the MCU *believes* about where things are.

That makes it the safest of the four to run. Where the preconditions are met it corrects the model; where they are not, it declines. It cannot leave the installation worse than it found it, which is not true of anything else in this chapter.

18.8 Mechanism 3 — Update network or controller from IMS

This mechanism treats the controller as authoritative and imports its configuration into the MCU. It is available at two scopes: for a single network from the network context menu, and for a whole controller from the Administration tab of its properties. Chapter 15 §§15.8 and 15.9 show where both are reached.

It runs in two stages. Product IDs are resolved first, so that the import knows which existing MCU objects correspond to which devices; the configuration is then brought across. One property of the second stage governs everything about how the mechanism should be used.

IT PRUNES, BUT IT CAN NEVER ADD

MCU objects that the controller does not know about are removed. Objects the controller has that the MCU lacks are not created — and rather than importing a partial model, the operation stops with an error and directs you to mechanism 4 to bring the missing objects in first.

The consequence is a rule worth memorising: **the MCU's object counts must never be fewer than the controller's before this is run.** Establish the population first — mechanism 4 for anything the MCU has never seen — and only then import the configuration on top of it.

What the pre-placed objects need is remarkably little: each requires only a Product ID that will match, or a valid short address. They may be generic placeholders, because the import sets type, manufacturer, role and settings from the controller. This is what makes building a floorplan off-site and integrating it with a live installation in one pass practical.

Stage 1 — resolving addresses

Product ID is tried first. Where it resolves, the object's short address is updated to match the physical device; where it does not, the short-address path is used instead. This is why a pre-placed object needs only one of the two — either route can carry it.

Entries that could not be resolved by Product ID are listed before the import proceeds, each tagged by source.

- **Physical** — an ID the controller reported that the MCU could not match to anything it holds.
- **MCU** — an ID configured on the floorplan with no physical counterpart on the bus.

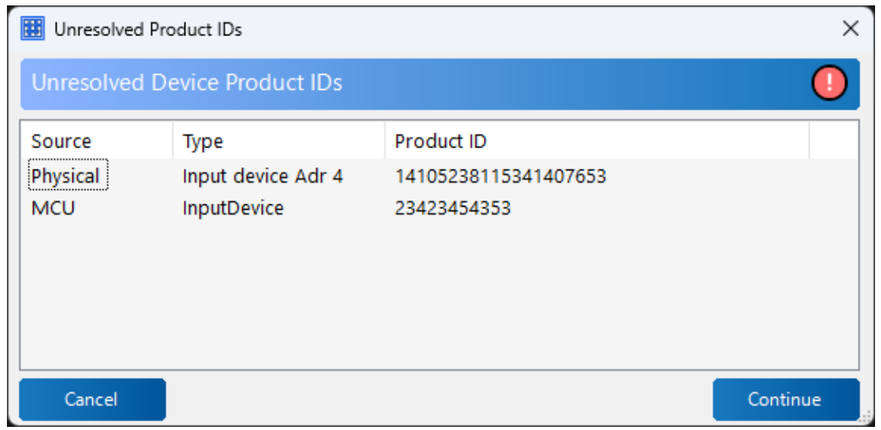


Figure 18.6. *The Unresolved Product IDs dialog. Source says which side reported the value that could not be paired.*

THIS LIST IS INFORMATIONAL — CONTINUE IS SAFE

Appearing here does *not* mean an object will be skipped. It means only that the product-ID route did not match; the object may still be resolved and imported by way of its short address. The dialog reports what could not be done the fast way, not what is about to be left out.

It is still worth reading. A long Physical list on a network you believed was fully described usually means the MCU is missing objects; a long MCU list usually means Product IDs have been entered that correspond to nothing installed.

Stage 2 — import confirmation

Once addressing is resolved the MCU asks for confirmation before importing. The dialog names the target and reports how many networks are affected — one for a network-scope run, or every network on the controller for a controller-scope one. That count is the quickest way to check you are about to do what you intended.

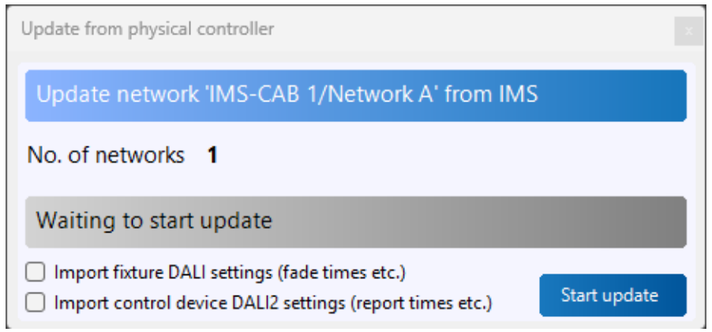


Figure 18.7. *The import confirmation. Both DALI-settings checkboxes are unchecked by default, and that default matters.*

- **Import fixture DALI settings** — fade times and the like.
- **Import control device DALI2 settings** — report times and the like.

LEAVE THE CHECKBOXES CLEAR UNLESS YOU MEAN IT

Left unchecked, the MCU ignores the controller's device-specific DALI values, and those properties fall through the normal inheritance chain instead (Chapter 5 §5.2). Checked, they arrive as explicit per-device overrides on every object imported.

The default is the right choice for most imports, and the reason is maintenance rather than fidelity. An installation configured device by device on the controller becomes, on import, an installation configured by inheritance in the MCU — where a change made at zone or building level actually propagates. Ticking the boxes preserves the old arrangement faithfully and, in doing so, pins several hundred values as overrides that will then quietly ignore everything set above them.

Tick them only where the per-device values are themselves the intent, rather than an artefact of how the controller happened to be commissioned.

18.9 Mechanism 4 — Scan network for new devices

Scanning asks the bus what is on it. It is the discovery mechanism, and the one to reach for when the MCU's model is missing devices rather than merely mistaken about where they are.

Discovery is keyed on the **GTIN** — the model identifier — and every discovered GTIN must have a matching entry in the global MCU catalogue (Chapter 27). The check runs **at scan time, before anything can be dragged**. If any discovered device carries a GTIN the catalogue does not hold, the scan stops with an error listing the unmatched numbers, and nothing can be imported until those models have been added. The MCU establishes what a device *is*, and therefore what it is capable of, before it will let anyone say where it belongs.

Discovered objects are then dragged from the Device Discovery window onto the floorplan, and the order in which they may be taken is enforced rather than merely recommended.

ZONES FIRST, AND THE SOFTWARE INSISTS

Zones must be imported before any fixtures or input devices. The Fixtures and Input Devices tabs are padlocked until every discovered zone has been imported, at which point the Zones tab shows a tick and the others unlock.

The reason is that references between objects have to survive the import. Dropping a zone automatically brings in the fixtures and input devices it uses, along with the links between them — so taking the devices first would strand exactly the relationships the scan exists to preserve.

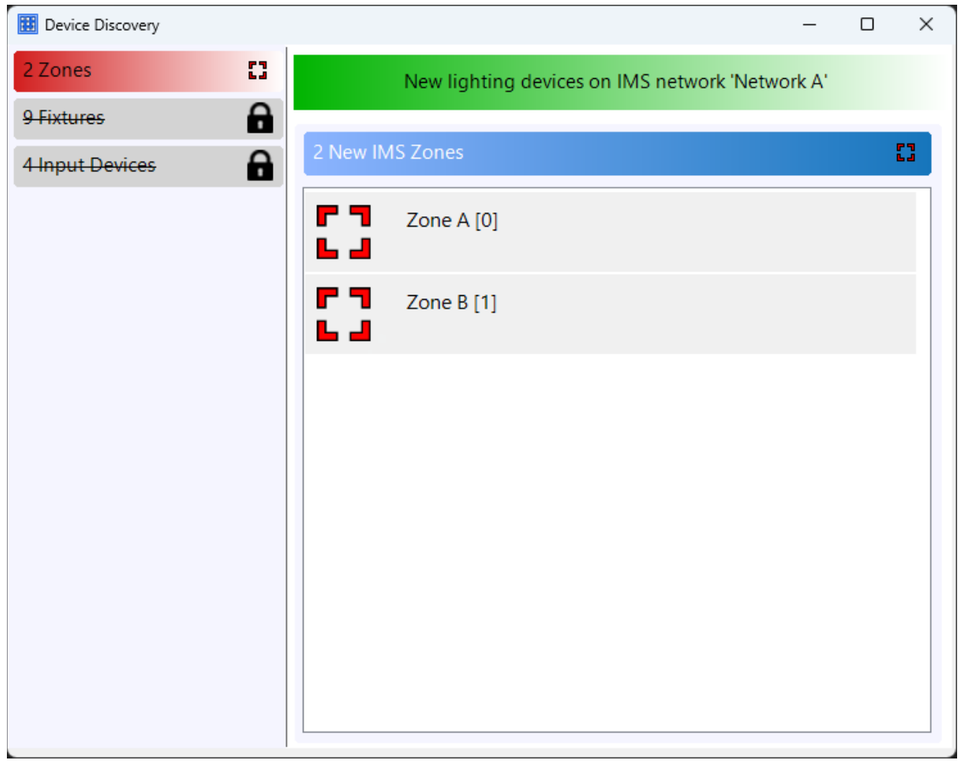


Figure 18.8. *Device Discovery on arrival. Nine fixtures and four input devices have been found, but both tabs are padlocked until the two zones are taken first.*

The window presents one tab per object type, each carrying a count, and lists the discovered items of whichever type is selected. Zones are matched by zone number, shown in brackets after the name — a reminder that the number is the DALI group rather than a label (Appendix C §C.3).

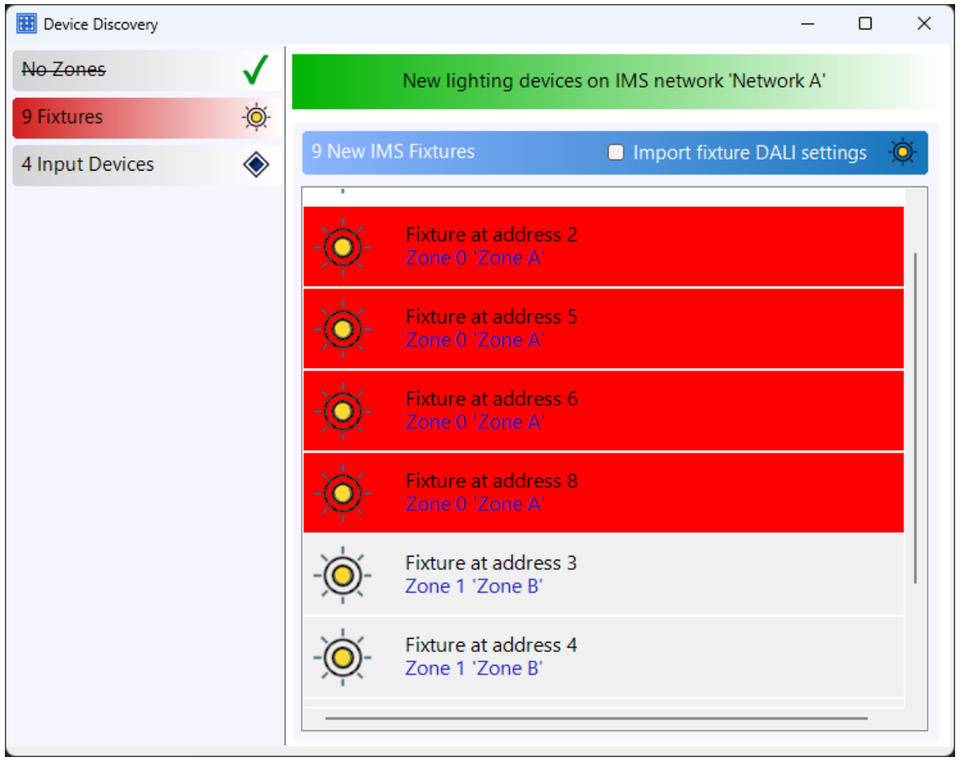


Figure 18.9. *The Fixtures tab once the zones are in. The Zones tab now shows a tick, and each discovered fixture carries its address and the zone it belongs to. The highlight is a multi-selection ready to drag.*

Items can be selected several at a time and dropped together; the MCU arranges them on the floorplan itself. Each discovered fixture shows both its physical address and the zone it came from, which is usually enough to tell at a glance whether the scan has found what you expected. A per-type Import DALI settings checkbox mirrors the one in mechanism 3, and the advice in §18.8 applies here too.

> **TWO THINGS THAT SAVE TIME**
>
> Dropping a discovered device also imports and resolves its physical short address. There is no separate addressing pass afterwards — a device that arrives by discovery arrives fully placed.
>
> Discovery need not be exhaustive in one sitting. You can import some objects now and re-scan later for the rest, which makes it reasonable to work through a large network in stages rather than committing to all of it at once.

18.10 Re-addressing the bus

The Addressing tab holds the two operations that change the physical addresses themselves rather than the MCU's record of them.

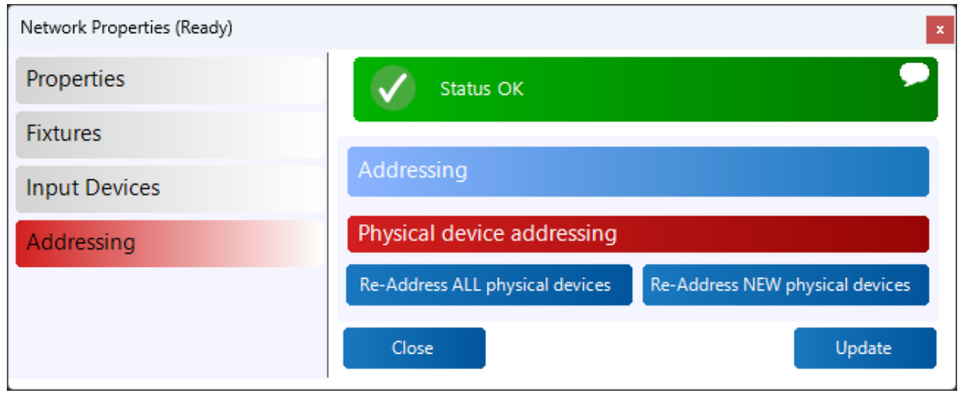


Figure 18.10. *The Addressing tab. The two buttons differ far more than their labels suggest.*

- **Re-Address NEW physical devices** — assigns addresses only to devices that do not currently have one. This is the ordinary operation after adding hardware, and it leaves every existing assignment untouched.
- **Re-Address ALL physical devices** — discards the entire existing assignment and runs the randomisation procedure across the whole bus from scratch. Appendix C §C.9 describes what that involves.

> **RE-ADDRESSING EVERYTHING IS NOT A REPAIR**
>
> Re-Address ALL invalidates every mapping the MCU holds for that network at a stroke, along with any relationship between an address and the fitting a person can point at. Scenes, zone membership and the work of any previous Identify & Assign run all lose their anchor.
>
> It is a commissioning operation, appropriate when a bus is being brought up for the first time or is in a state genuinely beyond reconciling. It is not a way of clearing a fault. When something is wrong, the mechanisms above exist precisely so that this button is not needed.

18.11 Working on a single device

Not every discrepancy is a network-wide problem. Where one device is wrong and the rest are fine, both fixtures and input devices carry their own addressing controls, and using them is far less disruptive than any network-scope operation. Chapter 17 §17.5 and §17.8 show those tabs; what matters here is when to reach for them.

When a single device refuses to pair during a product ID match, the place to look is that device's own Properties tab (Chapter 17 Figure 17.1). A record whose Product ID is blank or wrong cannot be matched to anything, and will present as a count mismatch that appears to make no sense — the counts look right, and yet the pairing fails.

Where an address simply needs correcting, the device's Addressing tab shows the addresses already in use on that network, so a conflict cannot be created by accident. Note that the two address spaces are separate, as Appendix C §C.3 explains: a fixture and an input device may both sit at address 6 on the same network without any conflict at all.

18.12 Choosing a mechanism

Collecting the chapter into one place. Find the situation, not the button.

Situation	What to reach for
A commissioned bus, and an MCU network that is empty	Scan to establish the population (4), then import configuration (3). Identify & Assign (1) for anything left ambiguous.
Hardware has been added	Re-Address NEW if the new devices have no address, then scan (4) and drag into zones.
Addresses have shifted; the population is unchanged	Update addresses from matching product IDs (2).
The controller is right and the MCU has drifted	Update from IMS (3) — but check the MCU's counts are not fewer first.
One device is wrong; the rest are fine	That device's own Addressing tab (§18.11).
Counts do not match and the cause is not clear	Do not run 2 or 3. Scan (4) to see what is actually there, then reconcile.
Nothing matches and nobody can account for it	Identify & Assign (1). Slow, but it settles the question.

> *Every mechanism here except one is an inference from data that might itself be wrong. When the inferences disagree, stop reasoning and go and look at the ceiling.*

PART III · FOR ENGINEERS

19 IMS ↔MCU Synchronization

Two independent models of an installation exist at once: the one the MCU holds centrally, and the one the controller holds on its own hardware. This chapter is about how they are kept in agreement — how each side notices that the other has changed, which way an update then flows, and what governs whether it is permitted to flow at all. Making the two models agree in the first place is reconciliation, and that is Chapter 18.

19.1 Two models of one installation

An IMS controller is designed to run autonomously. Once configured it executes its own schedules and occupancy logic entirely on its own hardware, and it carries on doing so whether or not the Gateway and DataManager above it are reachable. The MCU does not drive the lights moment to moment — the IMS does. What the MCU adds is central configuration, monitoring, and coordination across many controllers and many buildings.

For that arrangement to work, two models of the same installation have to be kept in agreement: the MCU's, held centrally and edited through the administration and site-management tools, and the controller's own, held on the unit itself. Keeping them in step divides into two quite separate concerns, and it is worth being clear about which is which.

- **Keeping configuration in step** — how a change on one side is noticed, and how the system decides whether an update is needed and in which direction it should travel. This is the revision model, and it is the subject of this chapter.
- **Reconciling the device model** — how the MCU's configured objects are matched to the real DALI devices the controller reports, and how an existing physical installation is imported into the MCU. That is Chapter 18.

The first assumes the two sides already agree about which devices exist. The second is what establishes that agreement. A system that has never been reconciled cannot usefully be synchronized, which is why commissioning runs in that order.

> **WHO THIS CHAPTER IS FOR**
>
> Synchronization is an engineering and administration topic. The operations described here sit behind engineering permissions, and in normal day-to-day use they never need to be touched — a correctly commissioned system stays in step on its own, and Chapter 11 describes what standard users see instead.

19.2 The revision model

Both the MCU and each physical controller maintain their own revision number for that controller's configuration. Whenever a side's configuration changes, its revision advances. Comparing the two numbers tells the Gateway which side is ahead, and therefore which way an update should flow — or that none is needed.

That single comparison is the whole mechanism. Everything else in this chapter is either where the numbers are displayed, how far a change propagates before the comparison is made, or what is permitted to happen once the answer is known.

Where revisions are shown

Two places, both on the Lighting Controller Properties window and both covered in Chapter 15.

- The **Properties** tab header carries a numeric badge: the MCU's current revision number for that controller (Chapter 15 §15.2).
- The **Synchronization** tab shows the pair side by side — **Physical controller data version**, reported directly by the

controller and shown as a dash when it is unreachable, and **MCU Controller data version**, the number the MCU holds (Chapter 15 §15.7).

The Synchronization tab is purely informational. It reports the comparison; it does not itself trigger anything.

Building and area revisions

Revisions are tracked at more than one level. Alongside the per-controller number, the MCU keeps area-level revision numbers — internal to the MCU and Gateway — recording when an area's configuration last changed and therefore needs re-checking against its controllers. The cascade is scoped deliberately, to avoid doing more work than a change warrants: a building-level change causes all of its areas to be re-checked, while an area-only change refreshes just that area.

> **NOTHING HAPPENS UNTIL THE AREA IS SAVED**
>
> Configuration changes take effect only once the area is saved back to the DataManager. That save is what advances the area revision; the Gateway then notices the new number, reloads that area, and resynchronizes the affected controllers and their schedules, leaving areas that did not change undisturbed.
>
> This is worth knowing when a change appears not to have taken. The usual explanation is not that synchronization has failed but that the save has not happened yet — the MCU is holding an edit that the Gateway has no reason to have noticed.

19.3 Controller status at a glance

A controller's icon on the floorplan encodes both its connection state and its synchronization state, so a mismatch is visible without opening anything. The icons themselves are illustrated in Appendix B §B.4; what each condition means for synchronization is below. The two arrow states are the ones that describe a revision difference and therefore a pending update.

Controller icon	Meaning
Connected	Green. Connected and in step.
Status unknown	Green with a question mark. No status has been received from the controller yet.
Busy	Blue. Connected, but performing initial setup or synchronization. Direct commands are unavailable until it finishes.
Disconnected	Red cross. The Gateway cannot reach the controller.
IMS newer	Red, arrow down. The controller holds changes the MCU does not. The MCU can be updated from the controller.
MCU newer	Red, arrow up. The MCU holds changes ready to push down.
Physical devices changed	Red. The two device and address models do not agree. This is a reconciliation problem, not a revision one: see Chapter 18.

The distinction between the last row and the two above it matters. A red arrow means the two sides agree about what exists and disagree only about how recently it was configured, which the revision model can resolve on its own. Plain red means they do not agree about what exists, and no amount of synchronization will fix that — it needs reconciling first.

19.4 The two-tier sync policy

How aggressively the MCU keeps a controller in step is governed by two independent settings that combine. The building-level setting establishes the default policy; the controller-level setting can restrict it further, but never loosen it.

Setting	Options and Effect
Building update setting *Applies to all controllers in building*	• Never • ***Update IMS only if the gateway (MCU) is newer** (Building default) • Update IMS if it differs from the MCU Gateway • Always. The policy default applied to every controller in the building.
Controller update setting *Individually selectable per controller*	• Disabled — the Gateway ignores this controller entirely. • ***Read only —** the Gateway connects and reads status but never writes controller properties; direct and quick-test commands still work. (Default for new added controllers) • Normal — full read and write synchronization.

Chapter 15 §15.5 covers where Network mode is set and Chapter 13 covers Update mode. The point here is that both are consulted: a building set to Always will still not write to a controller left in Read only.

> **READ-ONLY BY DEFAULT — AN IMPORTANT SAFEGUARD**
>
> A newly added controller defaults to Read only. This deliberately protects a controller that was fully configured standalone before it was ever added to the MCU: until an engineer has reviewed it and imported its configuration, the Gateway will not push anything down that could overwrite the existing on-controller setup.
>
> Note what Read only does *not* block. Status is still read, and direct and quick-test commands still function — so an engineer can verify that the right fittings respond before committing to anything. It is a write barrier on configuration, not a disconnection. Once the controller has been reconciled, switching it to Normal enables full two-way synchronization.

19.5 Forcing a resync

When a controller needs to be put back into a clean state, **Restart controller connection** — on the Administration tab of Controller Properties, Chapter 15 §15.8 — forces the Gateway to drop the connection. The controller then reconnects on its own and resynchronizes. It touches the connection rather than the configuration, which is what makes it a safe first response to a controller that is present but behaving oddly.

A related building-level option, Update controller time from gateway, keeps the controller's clock accurate. This matters more than it sounds: the IMS calculates its own sunrise and sunset times locally from the building's latitude and longitude, so a drifting clock quietly moves every astronomical schedule on that controller.

19.6 Reconciling the device model

The revision model decides when to update and which way. It assumes, though, that both sides already agree about which devices exist — that every MCU object is matched to a real device at a known short address. Until that is true, status and control cannot meaningfully be exchanged for that device at all, and the controller's icon shows plain red rather than an arrow.

Establishing that agreement is a separate job with its own tools: four reconciliation mechanisms, described in full in Chapter 18. In outline they divide by how much work is needed — resolving addresses only, resolving addresses and importing configuration, or discovering objects the MCU does not yet hold at all. Chapter 18 sets out which situation each is for, and the order in which to reach for them.

19.7 From commissioning to steady state

The two halves fit together into a commissioning path that does not require an engineer to build the entire MCU model on site. It is worth following once end to end, because it shows where reconciliation stops and synchronization takes over.

1. **Off-site**, build the MCU floorplan for the building using generic placeholder objects, keying each only by Product ID or short address. Chapter 18 §18.7 explains why so little detail is needed.
2. On site, add the controller. It arrives in **Read only** mode, protecting whatever configuration it already carries.
3. If the floorplan is missing anything the controller has, scan for new devices to discover and place it — zones first, then fixtures and input devices (Chapter 18 §18.8).
4. Run **Update controller from IMS** to resolve every address and import the full configuration in one pass. The MCU network now matches the controller.
5. Switch Network mode from Read only to **Normal**.

From this point the revision model takes over. Further changes on either side advance a revision number, the Gateway notices, and the two models are kept in step without anyone intervening — which is the quiet, hands-off state a correctly commissioned system spends most of its time in.

> *Reconciliation is what makes the two models agree. Revisions are what keep them agreeing. The first is a job with an end; the second runs for as long as the building does.*

PART III · FOR ENGINEERS

20 Scheduling

Occupancy decides what a space does moment to moment; the schedule decides what it is allowed to do at all. This chapter covers building a schedule — and, most importantly, the constraint model that makes scheduled events cooperate with occupancy rather than fight it.

20.1 How scheduling works here

Each **area** has its own schedule, made up of any number of events. This is worth pausing on, because it is not how lighting schedules usually work: an event does not *set* a zone to a state. It sets a **limit** — a floor, a ceiling, or both — within which the zone continues to behave normally.

So a schedule never overrides occupancy; it constrains the range occupancy may move within. During working hours you might allow the full range. Overnight you might impose a ceiling, so that a passing cleaner triggers a modest level rather than full working light. In a secure area you might impose a floor, so that a corridor never goes fully dark. The zone still responds to its sensors either way — it simply cannot leave the band the schedule permits.

> **The whole model in one sentence**
>
> Occupancy proposes; the schedule constrains; demand response trims. Each layer works on the result of the one before it, and none of them needs to know about the others.

20.2 The schedule calendar

Opening an area's schedule presents a month calendar. Days on which events are active are marked, and hovering one lists what is scheduled that day and the pattern it repeats on — a quick way to confirm that an event you have just built lands where you expected.

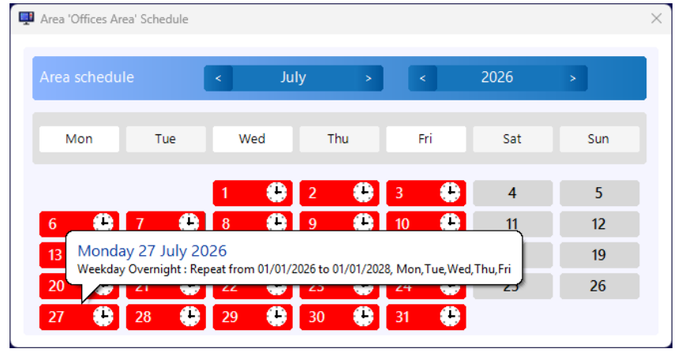


Figure 20.1. *The area schedule calendar. Marked days carry events; hovering shows what runs and when.*

20.3 The events list

Behind the calendar is the list of events themselves, which is where they are created and managed. Each row shows the event's **priority**, its name, when it is **next due**, a plain description of what it does, and its repeat pattern.

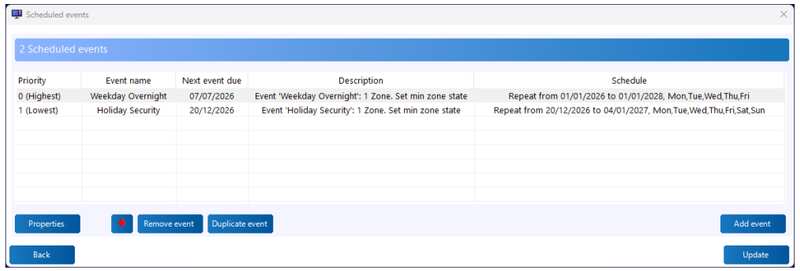


Figure 20.2. *The scheduled events list. Priority 0 is the highest.*

Alongside the usual add and remove actions is **Duplicate event**, which is the practical way to build a set of related events — create one carefully, then duplicate and adjust it rather than starting each from scratch.

20.4 Defining when an event runs

An event's first page defines its timing, in three layers that combine: an overall **date range** the event is active between, a **daily schedule** of when it starts and ends each day, and a **weekly pattern** of which days it applies to.

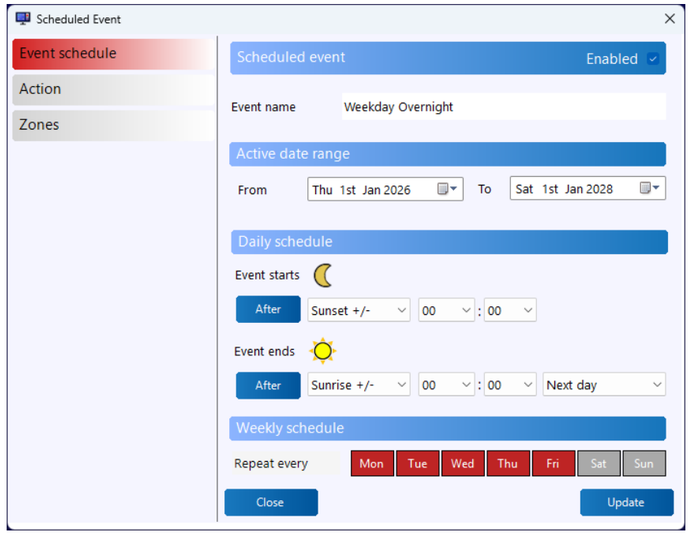


Figure 20.3. *Event timing: active date range, daily start and end, and the days it repeats on.*

The daily start and end are the interesting part. Each can be tied either to a **fixed time** or to **sunrise or sunset**, with an offset before or after — so "thirty minutes after sunset" is expressed directly, and follows the seasons automatically. Where an event runs past midnight, its end carries a **next day** qualifier.

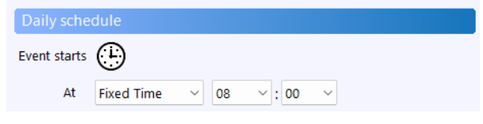


Figure 20.4. *The same control set to a fixed time instead of a solar event.*

> **Sunrise and sunset are calculated locally**
>
> The controllers work out sunrise and sunset themselves, from the latitude, longitude and timezone set on the building (Chapter 13). Nothing needs to fetch them, and they keep working with no connection to the gateway at all — but they are only as accurate as the building's recorded position, so it is worth confirming that when commissioning.

20.5 What an event does

The Action page is where the constraint is set. Two independent options are available, and an event may use either or both:

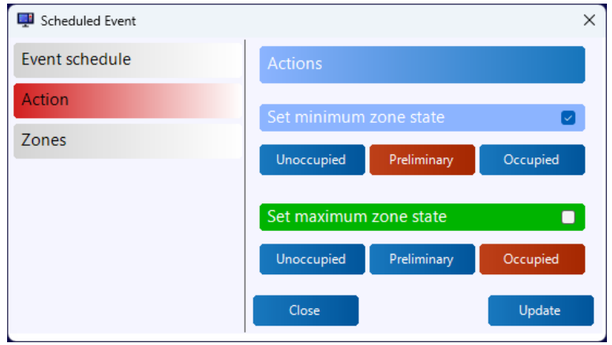


Figure 20.5. *Setting a minimum zone state — the floor below which affected zones may not fall.*

Action	Effect while the event is running
Set minimum zone state	A floor. The zone may not fall below the chosen state — useful for keeping a route lit overnight, or holding a background level in a public space.
Set maximum zone state	A ceiling. The zone may not rise above the chosen state — useful for preventing full working light outside working hours.

Setting both produces a band. An event that sets a minimum of Preliminary and a maximum of Preliminary pins the zone there; one that sets no minimum and a maximum of Preliminary lets the space fall dark when empty but prevents it reaching full brightness when someone passes through.

20.6 Which zones an event affects

The Zones page selects what the event applies to, as a tree of the area's controllers, their networks and the zones within them. Ticking a controller or a network selects everything beneath it, so "the whole area" and "just these three zones" are equally quick to express.

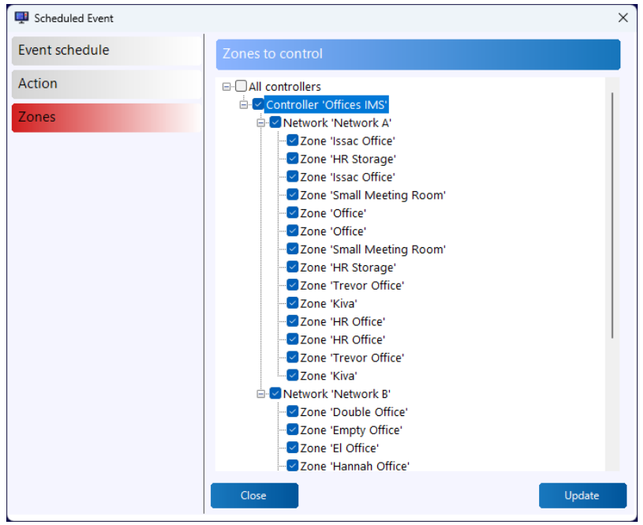


Figure 20.6. *Selecting the zones an event applies to.*

20.7 Priority and overlapping events

Events will overlap — a general weekday rule and a holiday exception may both cover the same evening. This is resolved by **priority**, where **0 is the highest** and larger numbers rank lower.

Resolution happens **per zone**, not per event. For each zone, the system considers every event currently running that names it and applies the constraint from the highest-priority one. Two events covering different zones never interact at all, even if they run at the same moment — so a building-wide overnight rule and a targeted exception for one wing coexist without either being disabled.

> **Designing a schedule**
>
> A workable pattern is to write the general rule at a low priority covering everything, then add exceptions at higher priorities covering less. A weekday overnight ceiling across the whole area at priority 1, and a holiday rule at priority 0 that takes precedence when it applies, is easier to reason about — and to change later — than a large set of mutually exclusive events.

20.8 Schedules and the controllers

Like everything else, a schedule is authored in the MCU and delivered to the controllers when the area is saved. From then on the controllers run it themselves — which is why scheduled lighting continues through a gateway outage, and why the sunrise calculations are done locally.

Each controller holds only the part of the schedule concerning its own devices, and the controllers do not interact. One consequence is useful during commissioning: an existing schedule can be **imported back up** from the controllers, the MCU assembling the separate pieces into a single area schedule. Chapter 19 covers that operation, which is intended for onboarding an existing installation rather than everyday use.

PART III · FOR ENGINEERS

21 Demand Response and Daylight Harvesting

Two mechanisms reduce the light a building uses without anybody asking them to. Demand response trims consumption on request, temporarily and across the whole site. Daylight harvesting gives back what the windows are already supplying, continuously and zone by zone. They work quite differently, but they share one property worth understanding before either is configured: both act on the result of the zone's own occupancy logic rather than replacing it.

21.1 Two ways to use less light

Chapter 4 describes how a zone decides what it wants: an occupancy state, a light method, a level. That decision is the starting point, not the final answer. Two further mechanisms sit above it, and each modifies the result in its own way.

- **Demand response** reduces output by a proportion, for as long as a reduction is being asked for. It is external in origin, building-wide in scope, and temporary.
- **Daylight harvesting** targets a measured illuminance rather than a level, adjusting continuously so that electric light makes up only the shortfall against what daylight is already providing. It is local, permanent, and driven by a sensor in the space.

> **NEITHER OF THEM TURNS LIGHTS ON**
>
> Both mechanisms are reductions. Demand response dims what would otherwise be brighter; daylight harvesting supplies less because the sun is supplying more. Neither will bring a zone up, and neither will hold light in a space its occupancy logic believes is empty.
>
> This is why the manual keeps returning to one sentence: *occupancy proposes; the schedule constrains; demand response trims.* Each layer acts on the result of the one before it, and every layer after the first can only take away.

21.2 What demand response is for

Demand response is a request from outside the building to use less electricity for a while — from a grid operator during a peak, from a building management system balancing load, or from a peak demand utility billing schedule that makes certain hours expensive. Lighting is a useful thing to trim, because a modest reduction across a whole site adds up while being close to imperceptible in any one room.

Demand response is license-gated, so where a client's licensing does not include it the feature is unavailable rather than merely idle (Chapter 25). Where it is available, the MCU expresses it as three levels of increasing severity. The levels themselves carry no fixed meaning — what matters is that they are ordered, and that each zone declares for itself how much it is prepared to give up at each one.

21.3 Setting what a zone gives up

The response is configured as three reductions, one per level, and it follows the ordinary inheritance chain: building, then zone type, then zone (Chapter 5 §5.2). The top of that chain is the natural place to set it.

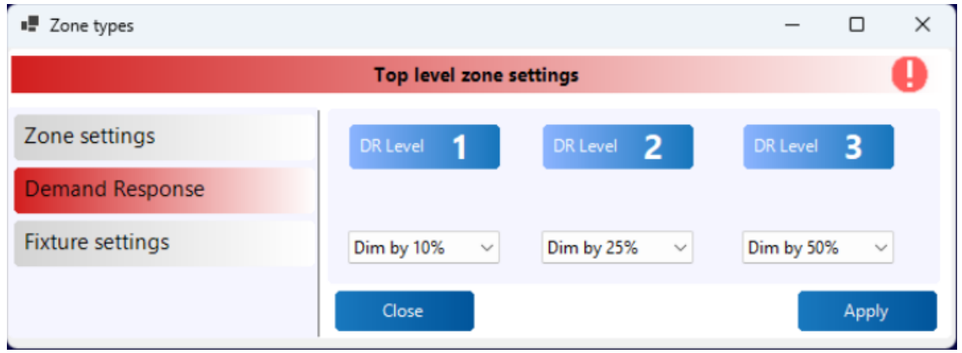


Figure 21.1. *Demand response on a zone type, setting the reduction applied at each of the three levels.*

Each level takes a proportional reduction — dim by ten per cent, by twenty-five, by fifty. Because the figure is a proportion rather than a level, it applies to whatever the zone would otherwise have been doing: a zone already sitting low gives up less in absolute terms than one at full output.

The dropdown spans the full range:

- **No action** at one end — a reduction of nothing, and the way a zone is excluded from a level entirely.
- **Turn Off** at the other — the full hundred per cent. A real option, and the right one for spaces that can simply go dark under genuine pressure.

> **WHY THE REDUCTION IS PROPORTIONAL RATHER THAN ABSOLUTE**
>
> A flat absolute reduction would be close to invisible in a brightly lit space and glaring in a dim one, because the eye is far more sensitive to a change at low levels than at high ones. Taking twenty units off a zone at full output is barely noticed; taking the same twenty off a zone already running low is obvious.
>
> That is the same fact that makes the DALI dimming scale logarithmic in the first place (Appendix C §C.5). Expressing demand response as a proportion works with that grain rather than against it: every zone gives up a comparable amount of *perceived* light, whatever level it happened to be sitting at when the request arrived.

The same three settings appear on an individual zone, where they can be overridden.

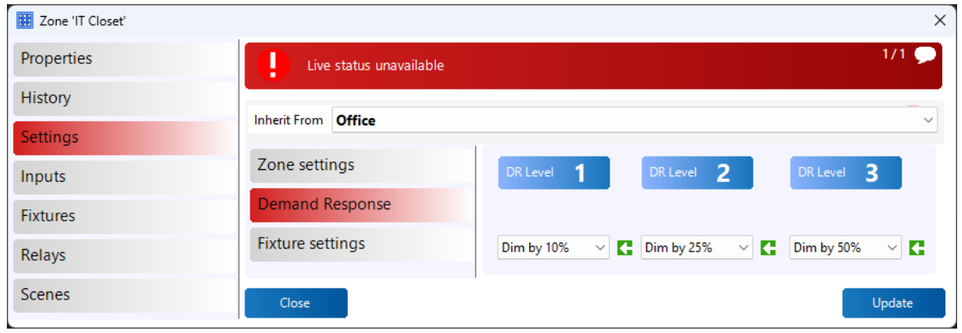


Figure 21.2. *The same settings on a zone, inheriting from the zone type 'Office'. The green arrows mark values set explicitly here rather than*

inherited.

> **SET THIS AT THE ZONE TYPE**
>
> Demand response policy is a judgement about which kinds of space can afford to lose light, and that judgement is the same for every office, every corridor and every plant room in the building. It is a property of the kind of space, not of the individual room.
>
> Setting it per zone means making the same decision two hundred times and maintaining it in two hundred places. Setting it on the zone type means making it once per kind of space, and overriding only where a particular room genuinely differs — a server room that happens to be classed as an office, say, or a reception that must stay presentable at any level.

A zone that must not dim at all is set to No action at all three levels. Because the levels are independent, a space can also be made to participate only under real pressure — No action at levels 1 and 2, a reduction at level 3 — which is often the right treatment for somewhere that matters but is not critical.

21.4 Driving demand response

The current state is held at the building, along with the controls for setting it.

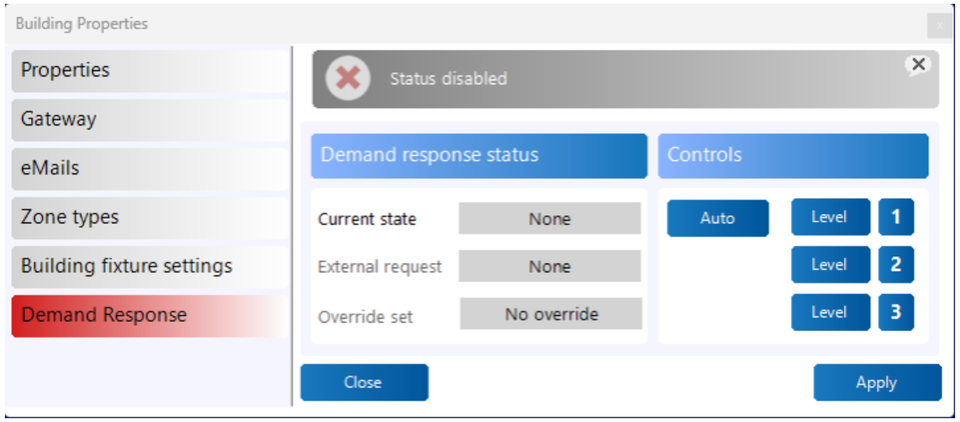


Figure 21.3. *Demand response at the building. The three status rows describe the situation; the buttons on the right decide what is done about it.*

The banner across the top of the figure reports that status is disabled, and that is not a demand response setting. It reflects the controller beneath, whose own status was disabled when the capture was taken (Chapter 15 §15.5). With no live status coming up from the controller there is nothing to report about what demand response is currently doing — the settings remain visible and editable; what is missing is the live state.

The three status rows answer three different questions, and reading them together is what tells you why a building is dimmer than expected.

Row	The question it answers
Current state	What level is actually in force right now.
External request	What is being asked for from outside.
Override set	Whether somebody has intervened manually, and at what level.

The controls beside them set the last of these. Auto returns the building to following the external request; the three level

buttons impose a level directly regardless of what is or is not being asked for. That makes them useful for two quite different jobs: forcing a reduction the grid has not asked for, and — more commonly — proving during commissioning that each level does what the zone types say it should, without waiting for a real event.

> **READ THE ROWS BEFORE CHANGING ANYTHING**
>
> A building sitting at level 2 with no external request is being held there by an override, and the fix is Auto rather than anything in the zone configuration. A building at no level despite an external request is either not licensed for demand response or has been overridden to Auto-with-nothing-asked. The rows distinguish these cases; the zone settings cannot.
>
> This is worth checking first when a site reports that lights are unexpectedly dim, because the alternative — working through zone types looking for a misconfigured reduction — finds nothing when the cause is an override somebody left set.

21.5 What daylight harvesting is

A zone whose light method is **Light levels** does not hold a fixed output. It holds a *target illuminance*, expressed in lux/fc, and adjusts its fixtures continuously so that the light actually measured in the space stays near that target. When daylight rises the electric contribution falls; when cloud comes over, it rises again. Chapter 16 covers setting the method and the target for each zone state.

The distinction between a target and a level is the whole idea, and it is easy to lose. A zone set to a level is being told how hard to work. A zone set to a target is being told what result to achieve, and how hard it works is whatever that takes at the time. The first is a number you can predict; the second is a number that moves all day.

> **IT NEEDS A PHOTOSENSOR, AND IT NEEDS THE RIGHT ONE**
>
> The target can only be held if something is measuring the space. A zone with a light method of Light levels and no working photosensor input has no feedback to act on — Chapter 17 §17.6 covers how sensor instances are associated with a zone, and Chapter 5 §5.1 how a missing one is reported.
>
> Where the sensor sees matters as much as whether it exists. A sensor with a strong view of the fixtures it is controlling measures its own output more than the daylight, which makes the loop harder to settle and the space darker than intended near the window.

21.6 Tuning the loop

Daylight harvesting is a feedback loop in which the controller changes the very thing its sensor is measuring. Left undamped, that loop hunts: the controller dims, the sensor reports a darker room, the controller brightens, and occupants watch the lights breathe. Three settings at three different levels exist to prevent it.

Setting	Where it lives
Adjustment interval	The controller (Chapter 15 §15.6). How often measured light is re-evaluated against the target.
Target tolerance	The controller (Chapter 15 §15.6). How far from target the measurement may drift before anything is done about it.
Hysteresis	The sensor instance (Chapter 17 §17.7). How far the measurement must move before the instance reports a change at all.

> **WIDEN THE TOLERANCE BEFORE YOU LENGTHEN THE INTERVAL**
>
> Both settings will stop a loop hunting, but they do it differently. Widening the target tolerance addresses the cause: it tells the controller that a room slightly off target is close enough, which is true, and which stops the corrections being made at all.
>
> Lengthening the adjustment interval only makes the same corrections less often. The hunting continues; it is merely slower, and it becomes more visible when it happens, because each correction is now larger. A room that adjusts smoothly every thirty seconds is less noticeable than one that lurches every five minutes.
>
> The sensor's own hysteresis is the third lever and the bluntest, because it suppresses reporting rather than acting. Reach for it last.

21.7 How the layers combine

Both mechanisms in this chapter sit above the zone's occupancy decision, and both can only reduce. Putting the whole arrangement in order:

1. **Occupancy proposes.** The zone's state machine decides what the space needs — a level, or a target (Chapter 4).
2. **The schedule constrains.** Time of day may permit less than occupancy asked for (Chapter 20).
3. **Daylight harvesting supplies the shortfall.** Where the method is Light levels, the electric contribution is whatever is needed to reach the target given what the windows are already providing.
4. **Demand response trims.** A proportional reduction is applied on top, for as long as a level is in force.

Each layer acts on the result of the one before it. This ordering is what makes the behavior predictable: no layer needs to know what the others are doing, because each simply reduces what it was handed.

> *The IMS runs all of this locally. A demand response level in force, a daylight target being held, a zone counting down its timeout — none of it depends on the gateway, the DataManager, or anyone being logged in.*

21.8 External demand response (OpenADR)

Everything so far in this chapter has been demand response the building decides for itself — levels configured per zone type, triggered and overridden locally. **External demand response** is the other direction: a signal from outside the building, from a utility or an aggregator, asking it to reduce load during a grid event. The MCU speaks the **OpenADR 3.1** protocol to receive those signals, and maps them onto the same three reduction levels the rest of the chapter describes.

In OpenADR terms the utility runs a **VTN**, the server that publishes events, and the building is a **VEN**, the client that listens and acts. The gateway's service for a building takes the VEN role: it connects to the VTN, follows a program of events, and drives the building's demand-response level as those events dictate. It is configured on the **ADR configuration** entry of Building Properties (Chapter 13), present only when the client's license includes ADR.

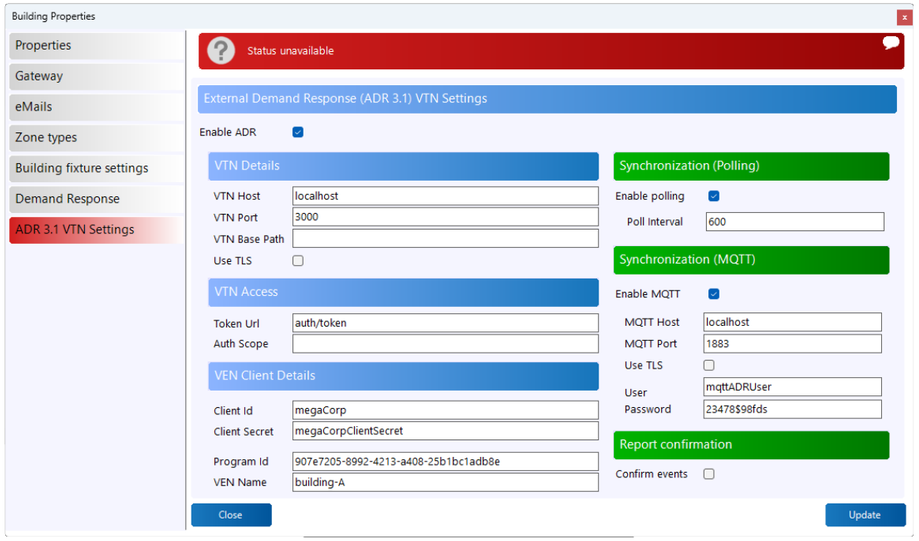


Figure 21.4. *The ADR 3.1 VTN settings on Building Properties. The four groups — VTN details and access, VEN client identity, the two notification transports, and report confirmation — are documented field by field in Appendix D.*

The screen carries a good deal of connection detail — host and port, OAuth credentials, MQTT settings — and the field-by-field reference lives in **Appendix D**. What matters here is the behavior those fields produce.

An OpenADR event carries a **SIMPLE** signal, a value from 0 to 3, which the VEN maps directly onto the building's demand-response levels: 0 is normal, and 1, 2 and 3 are the three reductions set per zone type in §21.3. When the VTN publishes an event raising the level, the building applies the matching reduction; when it ends, the building returns to normal.

> **EXTERNAL EVENTS ACT ONLY IN AUTOMATIC MODE**
>
> An incoming ADR signal drives the building only while its demand response is in **automatic mode**. A manual override set on the building (§21.4) takes precedence: the ADR client goes on receiving events, but the zones follow the override until it is cleared with Auto.
>
> This is the same Override-set mechanism described earlier — an external signal is simply another thing automatic mode responds to, and the same manual control sits above it. A building unexpectedly ignoring a grid event is very often one left in override.

The VEN learns of events two ways, and can use both at once: it **polls** the VTN on a timer, and where the VTN supports it, receives **MQTT** push notifications. Polling keeps the building in step through an event that steps between levels over time; MQTT makes it notice a new event promptly. Appendix D covers why both are usually enabled, and why the poll interval must be short enough to catch the level changes within an event.

> External and internal demand response are not two systems. The grid signal chooses a level; the building's own configuration decides what that level *means* — which zones dim and by how much — exactly as for a level the building chose itself. OpenADR adds a new way to ask; it changes nothing about how the building answers.

> *The IMS runs all of this locally. A demand response level in force, a daylight target being held, a zone counting down its timeout — none of it depends on the gateway, the DataManager, or anyone being logged in.*

22 The NICOR Administration App

Part IV covers administering the platform rather than any one building. This first chapter maps the Administration App so that the chapters which follow have a reference to: what the app is for, how it is organized, and who can use it.

22.1 What the app is for

Site Manager, covered in Parts II and III, is concerned with *a building* — its floorplan, its zones, its lighting. The Administration App is concerned with everything *around* buildings: who the customers are, who may log in and what they may do, what has been licensed, which software versions are current, which gateways exist, and what kinds of physical device the system knows about.

It is used rarely compared with Site Manager — typically when setting up a new customer, onboarding a colleague, or adding a product to the catalog — but the things it controls underpin everything else.

22.2 How it is organized

The app opens on a menu with two branches, and the distinction between them is worth learning because it explains where to find things:

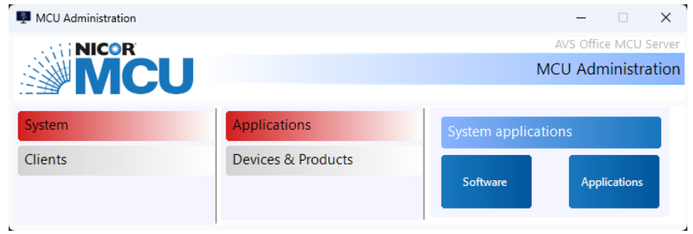


Figure 22.1. *The main menu. The System branch covers the platform; the Clients branch covers customers.*

Branch	Covers
System	The platform itself, shared by every customer: the software packages available for update, the registered gateway services, and the global catalog of device types the MCU can work with.
Clients	The customers and their people: user accounts and permissions, the location and building structure, client details and licensing.

Put simply: **System** is what the MCU knows; **Clients** is who it serves.

22.3 The System branch

Under System, **Applications** holds two related things — the installer packages published for the various MCU applications and services, and the registered gateway services themselves. Both are covered in Chapter 26.

Devices & Products is the global device catalog: the manufacturers, fixture driver types, lamp load types and input device types that the MCU understands. Every physical device discovered on a DALI network must have a matching entry here, so the catalog is what allows the system to recognize hardware at all. It also carries import and export, for moving catalog definitions between installations. Chapter 27 covers it in detail.

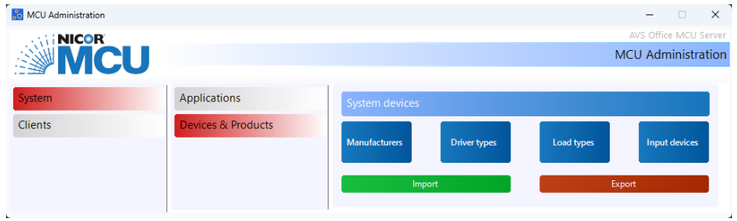


Figure 22.2. *The device catalog — manufacturers, driver types, load types and input devices, with import and export.*

22.4 The Clients branch

Under Clients are three areas: **Users** for accounts and permissions (Chapter 23), **Sites & Buildings** for the location and building structure (Chapter 24), and **Client details** for the customer record itself — which is also the route to their licenses (Chapter 25).

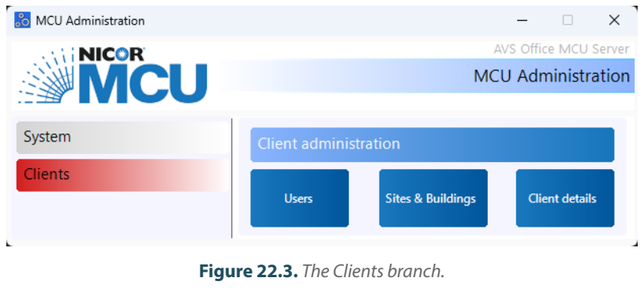


Figure 22.3. *The Clients branch.*

22.5 Who can use it

Access to the Administration App requires the **Administration** permission described in Chapter 11. Beyond that, what an administrator can see follows the same site-access rules as everyone else: an administrator attached to a particular client works within that client, while a **Root** administrator sits above all of them and can see every client on the installation.

> **Administrators cannot grant themselves more than they have**
>
> The system deliberately prevents privilege escalation: an administrator cannot give an account permissions beyond those they hold themselves, nor grant access to parts of the hierarchy they cannot reach. This means a client-level administrator can manage their own organization freely without ever being able to reach another customer's data — which is what makes it safe to delegate user administration to customers themselves.

23 Users and Permissions

Chapter 11 described permissions from a user's point of view. This chapter is the other side of that: how accounts are created, what each permission grants, and how to place a user in the hierarchy so they see exactly what they should.

23.1 The user list

Users are listed as a tree, grouped by the client they belong to. A **Root** branch sits above the clients for accounts that administer the installation as a whole; beneath it, each client holds its own users.

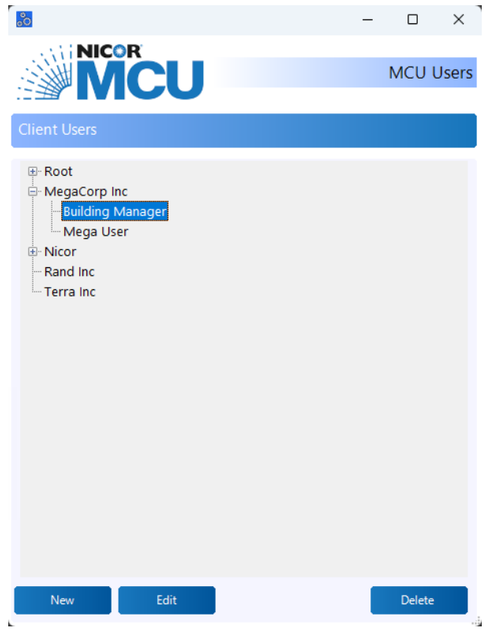


Figure 23.1. *Users grouped by client, with Root above them.*

Which clients appear depends on your own access. A Root administrator sees every client on the installation; an administrator belonging to a client sees only their own — which is what makes it safe to let customers manage their own people.

23.2 Creating and editing an account

The user dialog is in three parts: the person's details, the permissions they hold, and where in the hierarchy they sit.



Figure 23.2. *The user dialog: details, permissions, and site access.*

The details are straightforward — username, full name and email address, with a separate action for setting the password. The two parts below it are where the thinking goes.

23.3 Granting permissions

Permissions are granted individually rather than as a role, so an account holds exactly the set it needs. Chapter 11 explains what each one means in practice; the table below is a reminder of what you are granting when you tick each box.

Permission	Grants
Site Navigation	Browsing downwards from the account's assigned point in the hierarchy. Without it the account is confined to the level it was assigned.
Site Management	Use of Site Manager — opening buildings, viewing floorplans and reports. The everyday operating permission.
Modify Building	Making changes and driving lighting directly, including the quick-test controls. Without it a building is read-only.
Engineering	Commissioning and configuration — controllers, networks, addressing and the low-level engineering tools.
Administration	Access to this application — users, clients, licensing and the platform itself.

Three combinations cover most people. A building manager holds Site Navigation, Site Management and Modify Building. A commissioning engineer adds Engineering. A platform administrator adds Administration. Anything narrower is usually a deliberate restriction rather than an oversight — for example, withholding Modify Building from someone who should watch a building but not operate it.

23.4 Placing the account in the hierarchy

The lower half of the dialog sets **site access** — the point in the client's hierarchy this account is attached to, shown beneath the tree as its top-level access. An account may be placed at the client itself, at a location, or at a single building.

> **Site access and Site Navigation work together**
>
> Site access sets **where the account starts**; the Site Navigation permission decides **whether it can move down from there**. The combination is what gives fine-grained control:

Assigned at the client, *with* Site Navigation — the account can reach every location and building the client owns.

- Assigned at one building, *without* Site Navigation — the account is confined to that building and cannot see anything else.
- Assigned at a location, *with* Site Navigation — the account can work across that region but no further.

This is the mechanism behind most "why can that person not see the building?" questions. If an account is missing a building it should have, check its assigned point first and its Site Navigation permission second.

23.5 Escalation is prevented

An administrator cannot grant an account more than they themselves hold — neither permissions they lack, nor access to parts of the hierarchy they cannot reach. The restriction is structural rather than advisory, and it has a useful consequence: administration can be delegated to a customer without any risk that they reach beyond their own organization, or quietly promote themselves.

It also means an administrator cannot lock themselves out of the ability to fix a mistake by granting away something they need. Where a permission genuinely must be added beyond an administrator's own level, it has to come from an account above them — in practice, a Root administrator.

PART IV · FOR SYSTEM ADMINISTRATORS

24 Clients, Locations and Buildings

This chapter covers the structure everything else hangs from: the customer record, the geographic hierarchy beneath it, and the buildings within that. It is the first thing set up for a new customer and the thing user access is measured against.

24.1 The client record

A **client** is a customer of the installation, and the boundary that separates one customer's data from another's. The record itself is deliberately minimal — an identifier assigned by the system and a name — because its significance is structural rather than informational.

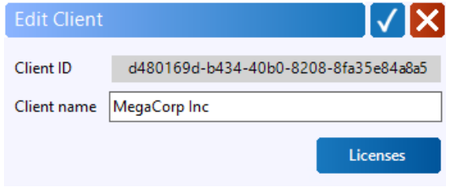


Figure 24.1. *The client record, with the route to its licenses.*

The record is also the route to the client's **licenses**, which are always held at this level. Chapter 25 covers them.

24.2 Locations

Beneath each client sits a tree of **locations** — a geographic structure of whatever depth suits the customer. A small organization might have a single level; a larger one might nest region, country and city. Locations exist to organize buildings and to give user access something meaningful to attach to.

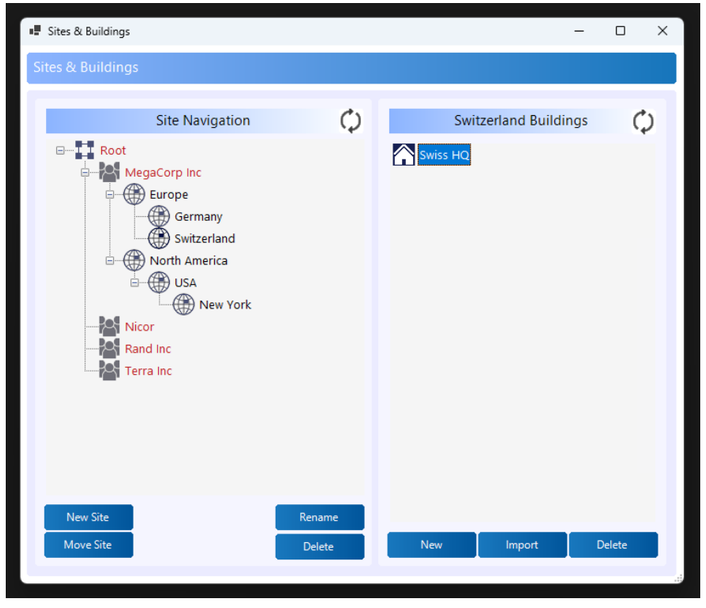


Figure 24.2. *Locations on the left, the selected location's buildings on the right.*

Locations can be created, renamed, moved elsewhere in the tree, and deleted. **Move** is the one worth knowing about — reorganizing a customer's structure as it grows is a normal thing to need, and moving a location takes its buildings with it.

> **Structure affects access**
>
> Because user accounts are attached to a point in this hierarchy (Chapter 23), the shape you give it determines what access you can express. A structure that groups buildings the way responsibility is actually divided — by region, by campus, by tenant — makes it straightforward to give each person exactly the right reach. A flat structure leaves only "everything" or "one building".

24.3 Buildings

Selecting a location lists its buildings in the right-hand pane, where they can be added, imported and removed. Creating one asks only for a name; everything else — position, gateway, areas, floorplans — is configured afterwards in Site Manager, as described in Chapter 13.

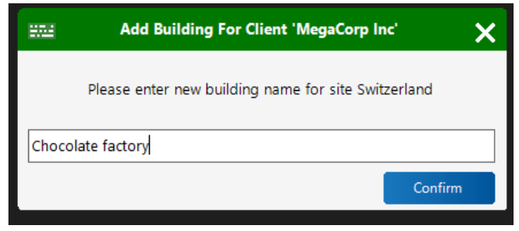


Figure 24.3. *Creating a building — only a name is needed at this stage.*

24.4 Importing and exporting buildings

Alongside creating a building from scratch, one can be **imported**. This is what makes it practical to build a configuration somewhere other than where it will finally live — preparing a building offline and bringing it into the live installation when it is ready, or copying a proven configuration to a similar site as a starting point rather than rebuilding it.

It pairs naturally with the off-site commissioning approach in Chapter 12: a building's floor plan and zone design can be prepared in advance, imported here, and then bound to real hardware on site.

24.5 Setting up a new customer

Putting the pieces together, a new customer is set up in roughly this order:

- Create the **client**.
- Add the **licenses** they have purchased — at minimum a gateway license, without which their gateways cannot connect.
- Build the **location** structure.
- Create or import the **buildings**.
- Create the **user accounts**, attaching each to the right point in the hierarchy.

From there the work moves to Site Manager and the commissioning sequence in Chapter 12.

PART IV · FOR SYSTEM ADMINISTRATORS

25 Licensing

Licenses decide which parts of the MCU a customer can use. This chapter explains how they are structured, what each one enables, and how to apply them.

25.1 How licensing is structured

Licenses are issued **per client**, and are held on the client record rather than against any building, gateway or user. A customer's licenses therefore apply across everything they own.

A license takes the form of a **key**, and each key grants one or more **permissions** — the individual capabilities it unlocks — together with an expiry date, which may be a fixed date or never. A client may hold **several keys at once**, and their permissions combine: one key might cover everything, or capabilities might arrive over time as separate keys as a customer adds them.

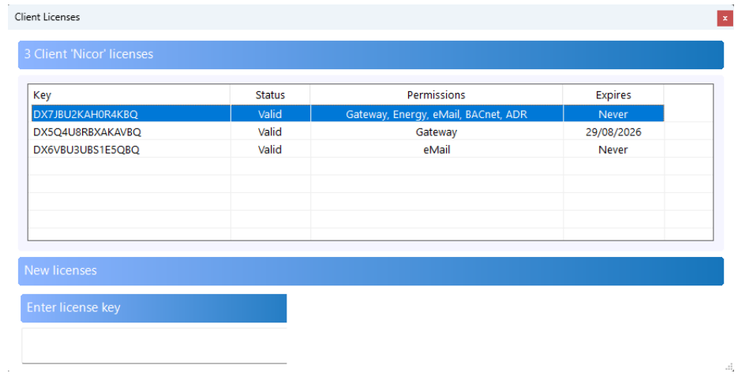


Figure 25.1. *A client's licenses. Each key lists the permissions it grants and when it expires; the permissions of all valid keys combine.*

25.2 What the permissions enable

The permissions carried by a key map onto the optional capabilities described elsewhere in this manual:

Permission	Enables
Gateway	Gateway connectivity. Without it the client's gateways cannot connect, and nothing else in the system functions — this is the foundational license.
Energy	Energy reporting — the historical consumption reports described in Chapter 10.
eMail	Email notifications — the building summaries and logs configured in Chapter 13 and at gateway level in Chapter 29.
BACnet	BACnet integration, allowing a building management system to see and drive the lighting. Covered in Appendix A.
ADR	Automated demand response — accepting demand-response requests from outside the system rather than only from a manual override.

> **Gateway first**
>
> Of these, the **Gateway** permission is the one to get right before anything else. It is easy to overlook during commissioning, and its absence presents as a gateway that will not connect — which is readily mistaken for a network or certificate problem. If a newly built system cannot establish its gateway connection, check the license before investigating anything more complicated.

25.3 Applying a license

New keys are added by entering them in the field beneath the list. Once accepted, the key appears in the grid with the permissions it grants, its status, and its expiry — and the capabilities it unlocks become available to that client's buildings.

Because permissions combine across keys, adding capability later is simply a matter of adding another key; there is no need to replace what is already there.

25.4 Status, expiry and when it takes effect

Each key shows a **status** and an **expiry**. Keys that never expire show accordingly; those with a fixed date need renewing before it passes if the capability is to continue.

What is worth understanding is *when a change takes effect*, because licenses are not evaluated continuously. There are two evaluation points:

Evaluated	What it governs
At user login	The capabilities available to that user's session. A license that lapses part-way through a session leaves it unaffected; the change appears at the next login, when the expired features become unavailable.
When a gateway loads a building	The gateway and BACnet capabilities. A running gateway carries on with what it already holds; a lapsed license takes effect the next time it needs to load a building.

This has two practical consequences. Applying a new license is not always instant — if a capability does not appear immediately, logging out and back in is usually all that is needed. And an expired license may not announce itself straight away: a system can keep running for some time on licenses that have already lapsed, only revealing the problem when someone next logs in or a gateway next reloads. Reviewing the grid periodically is therefore more reliable than waiting for a failure, particularly where a customer holds several keys with different dates.

> **When a feature vanishes for everybody**
>
> Licensing is a common explanation for a capability that has stopped being available to *all* of a customer's users at once — and the delayed evaluation above means it often appears the morning after the license actually expired, rather than at the moment it did. Permissions (Chapter 23) explain why one person cannot do something; licensing explains why nobody can.

PART IV · FOR SYSTEM ADMINISTRATORS

26 Software and Gateway Registration

The System › Applications area holds two things that sound similar but do quite different jobs: the software packages published for update, and the register of gateway services that exist on the installation.

26.1 Publishing software

The **Software** list holds the installer packages published for the MCU's applications and services. Each entry names the application it belongs to, carries a version, and holds the installer itself.

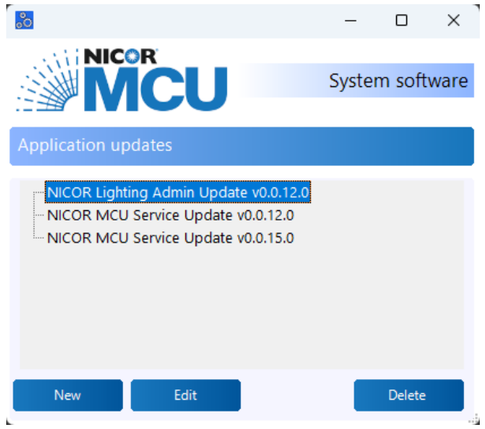


Figure 26.1. *Published software packages. Several versions of the same application may coexist.*

Several versions of the same application can be listed at once, which is what makes a controlled rollout possible — an older version remains published while a newer one is introduced.

Publishing a package means choosing the **application** it belongs to, giving it a **description** and a four-part **version number**, selecting the installer file, and — optionally — setting an **update-after** date and time.

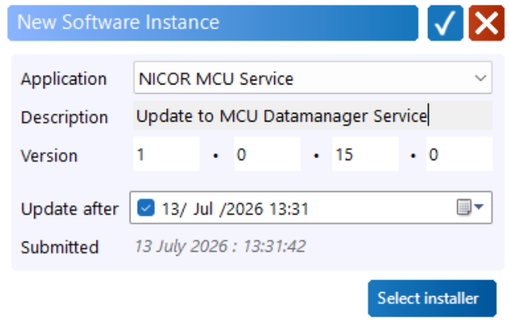


Figure 26.2. *Publishing a package: application, description, version, an optional update-after time, and the installer itself.*

> **Staging a rollout**
>
> The update-after time is what turns publishing into scheduling. A package can be prepared and uploaded during the working day but held back until an agreed window — out of hours, or after a change-control approval — without anyone needing to be present at the moment it becomes available.

26.2 Registering a gateway service

The **Applications** list is the register of gateway services known to the installation. Each entry is one running gateway — a site might have one, a large deployment several.

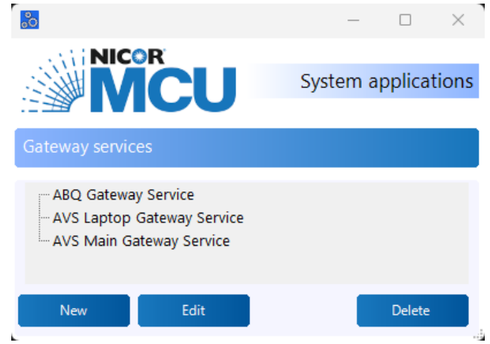


Figure 26.3. *The registered gateway services.*

Each registration records:

- The **application type** — which kind of service this is.
- An **instance identifier** — assigned by the system, and the thing that ties this record to the actual running service.
- A **description** — the name administrators will see when assigning it to a building, so it is worth making it meaningful.
- The **host** and its **HTTP and HTTPS ports** — where the service can be reached.

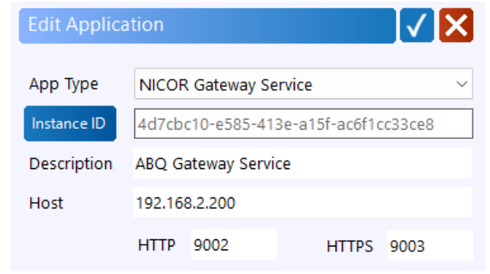


Figure 26.4. *A gateway registration — type, instance identifier, description and network address.*

26.3 How registration fits together

Registration is the MCU's *record* of a gateway; it does not by itself configure the service. Three pieces have to agree, and knowing which is which saves a good deal of confusion when a gateway will not connect:

Where	What it establishes
Here (registration)	That this gateway exists, what it is called, and where it lives on the network.

Where	What it establishes
Gateway Administration App	Which registered instance the running service actually is, plus its own local settings — covered in Chapter 29.
Building properties	Which gateway serves a given building — covered in Chapter 13.

The order matters during commissioning: register the gateway here first, so that it can be selected in the service's own administration app and then assigned to a building. Chapter 12 places this in the wider sequence.

27 The Device Catalog

The catalog is the MCU's knowledge of physical hardware: which products exist, how they behave, and how much power they draw. It is shared across the whole installation, and a great deal depends on it being right.

27.1 Why the catalog matters

Every DALI-2 device carries a **GTIN** identifying what it is, and when the MCU discovers devices on a network it looks each one up here. If a discovered device's GTIN has no catalogue entry, the discovery **stops** and reports the unmatched numbers — it cannot place a device it does not understand. Chapter 18 covers that operation; the point here is that the catalog is a prerequisite for it.

What the catalog supplies falls into three parts:

Entry	Describes
Fixture driver type	The control gear — how the device is addressed and driven, and what kinds of output it offers.
Fixture load type	The lamp — how much power it draws, and how that varies with dimming.
Input device type	A sensor or switch — and the individual instances it presents.

> **Driver and load are separate on purpose**
>
> A fixture in the MCU is a *driver* plus a *load*. The same driver may be fitted with different lamps, and the same lamp driven by different gear — so describing them separately means each combination does not need its own catalogue entry. It also puts the power characteristics where they belong: on the lamp, which is what actually consumes the electricity.

27.2 Manufacturers

Products are organized by manufacturers, so a manufacturer must exist before its products can be added. The record is simple — a name and a website address, with a button to open it.

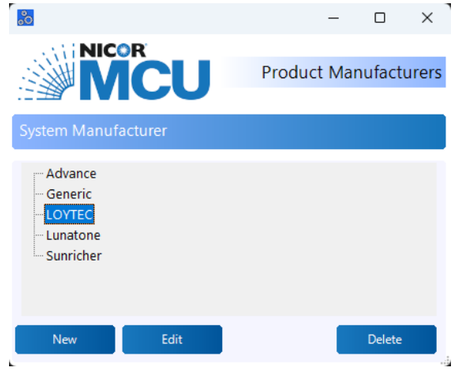


Figure 27.1. *The manufacturer list.*

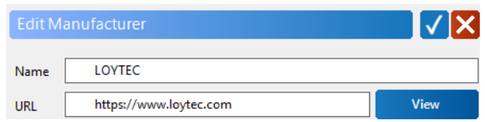


Figure 27.2. *A manufacturer record.*

A **Generic** manufacturer is useful for placeholder entries — particularly when building a floorplan off-site before the real hardware is known, as described in Chapter 12.

27.3 Fixture driver types

Driver types are listed as a tree beneath their manufacturers.

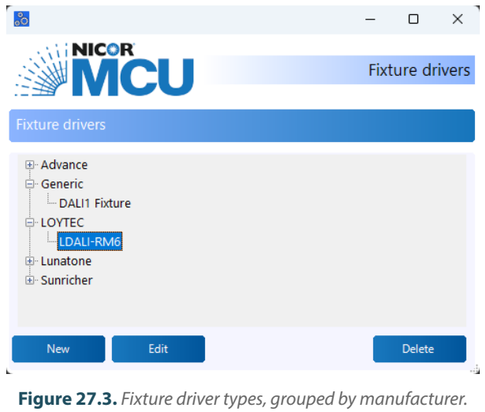


Figure 27.3. *Fixture driver types, grouped by manufacturer.*

A driver entry records how the device is identified and how it behaves:

- The **GTIN** — the identifier which discovery matches against. An optional additional identifier is available for cases where a GTIN alone does not distinguish one product from another.
- The **protocol** it speaks, and its **minimum and maximum arc levels** — the hardware limits within which every other setting must sit.
- Its **DALI bus current** in milliamps — what the device draws from the bus, which is what allows the system to report bus loading.
- The number of **supported devices**, for gear that presents more than one.
- Its **driver outputs** — whether it offers DALI dimming, a 0-10V output, a relay, or some combination.

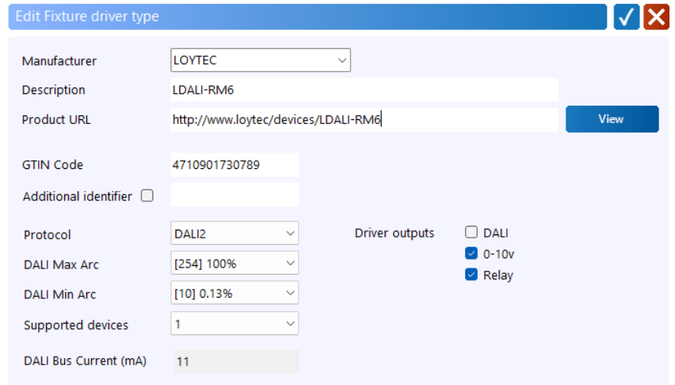


Figure 27.4. *A driver type: identification, arc limits, bus current and the outputs it offers.*

> **Outputs determine the roles a fixture can take**
>
> The driver outputs recorded here are what decide which **assigned roles** a fixture of this type can be given on the floorplan. A device offering a relay output can be used as a relay; one offering only DALI dimming cannot. Getting this right in the catalog is what makes the relay behavior described in Chapter 16 available where it should be — and unavailable where it would be meaningless.

27.4 Fixture load types and power

A load type describes the lamp itself, and exists almost entirely to answer one question: **how much power does this draw at a given dimming level?**

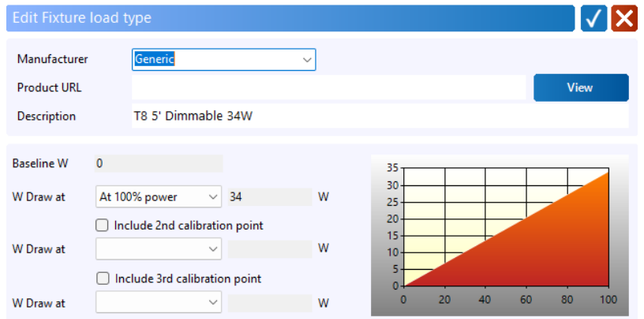


Figure 27.5. *A load type. The calibration points define a power curve, previewed alongside.*

It is defined by a baseline draw and the wattage at full output, with the option of adding second and third **calibration points** at intermediate levels. The graph alongside previews the resulting curve. A single point gives a straight line from baseline to full; adding points lets the curve follow a lamp that does not dim linearly, which most do not.

> **This is where every energy figure comes from**
>
> The MCU does not meter electricity. Every power and energy figure it reports — the live watts on the building view, and all of the historical reporting in Chapter 10 — is calculated from these curves against what each fixture was actually doing. The reports are therefore only as accurate as the load types behind them: a lamp cataloged with the wrong wattage will quietly skew every figure that includes it. Where accuracy matters, the calibration points are worth taking from the manufacturer's data rather than estimating.

27.5 Input device types

An input device entry records the same kind of identification as a driver — manufacturer, description, GTIN and DALI bus current — on its first page.

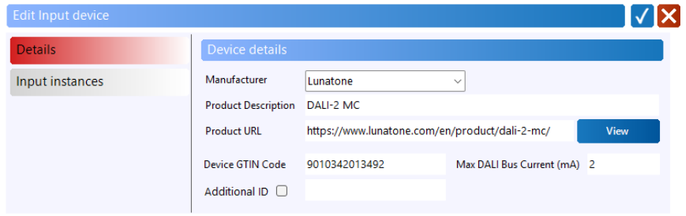


Figure 27.6. *An input device type.*

The second page is where the real substance is: the **instances** the device presents. A two-button switch declares two button instances; a multi-sensor declares its occupancy and light-level instances separately. Each instance carries a type, a number, a description, and its own settings.

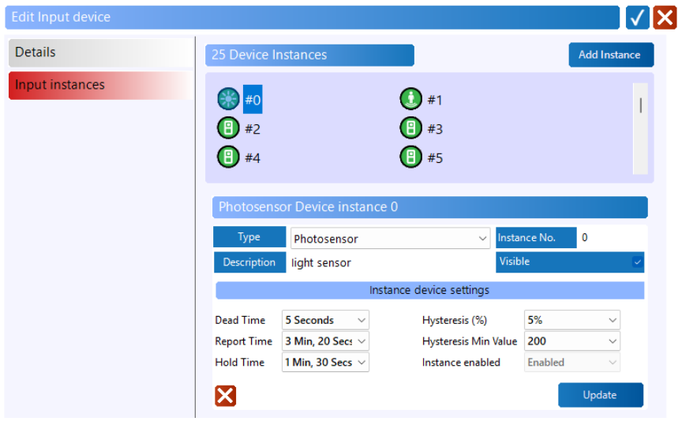


Figure 27.7. *The Input instances page. Every instance is listed by its type icon and number; the selected instance is shown below for editing.*

The page lists every instance the device declares, with the total in its header, each shown by an icon for its type and its instance number. Selecting an instance shows its current configuration below: its type, instance number and description, whether it is visible, and its device settings. Click **Update** to confirm changes to the selected instance, or the red **X** to remove it. New instances are added with **Add Instance**.

The **Visible** setting exists because some devices must declare instances for their own internal operation that serve no purpose in the MCU. An instance set to not visible still exists on the device, but it is hidden from the general interface: it cannot be selected as a zone input and does not appear in the device's properties.

The instance settings are DALI-2 behavioral parameters — among them:

- **Dead time** — how long the instance ignores further activity after reporting.
- **Report time** — how often it repeats a continuing condition.
- **Hold time** — how long a detected condition is held before being considered over.
- **Hysteresis** — for measuring instances such as light sensors, how much a reading must change before it is reported, which prevents a value hovering near a threshold from reporting constantly.
- **Hysteresis min value** — the smallest absolute change in a reading that counts, so that at low light levels a percentage hysteresis does not shrink to almost nothing.
- **Instance enabled** — whether the instance is active at all. A disabled instance stays declared but takes no part in control; Chapter 17 §17.7 covers using this to take a faulty input out of service.

> **Instance descriptions travel with the device**
>
> The description given to an instance here is what appears wherever that instance is used — including in a zone's Inputs list (Chapter 16). Naming them meaningfully in the catalog — "Top left button" rather than "Button 1" — pays off every time an engineer wires one to a zone.

27.6 Import and export

The catalogue can be exported and imported as a whole. This is how a curated set of product definitions is moved between installations — taking the products you have standardized on to a new deployment rather than re-entering them or distributing an updated catalogue to sites that need to recognize newly available hardware.

Because discovery depends entirely on the catalog, importing a well-maintained one before commissioning is a practical way to avoid the unmatched-GTIN interruption described at the start of this chapter.

PART IV · FOR SYSTEM ADMINISTRATORS

28 The Data Manager

The Data manager is the system's central data service —All MCU applications communicate directly with this service to read, write and synchronize their operations. This chapter covers its administration utility: setting it up, backing it up, and the storage options it offers.

28.1 What it holds

Everything the MCU knows lives here: clients, users and licenses; buildings, areas, zones and devices; schedules; the device catalog; historical energy data; and the audit log. Gateways load their buildings from it, and the administration applications read and write it directly.

Its central position is why the resilience behavior described in Chapter 2 matters. If the Data manager is unavailable, the applications cannot log in or load anything — but gateways that are already running carry on with what they hold, and the controllers below them are unaffected. Lighting does not stop; management does.

The service is administered through its own utility, which has two pages.

28.2 Settings

The Settings page covers the service itself, its identity, and the database it uses.

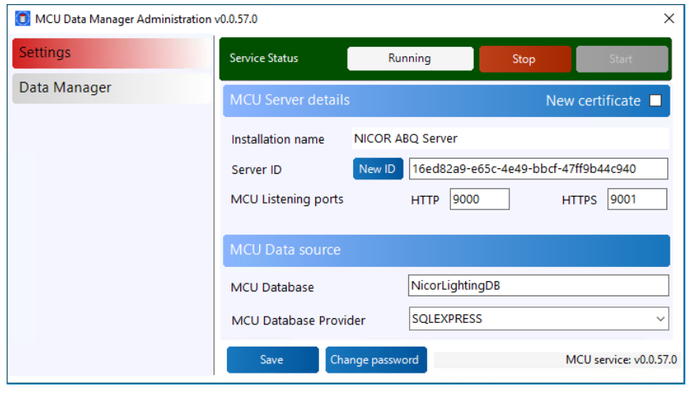


Figure 28.1. *The DataManager settings page.*

Setting	Purpose
Service status	Whether the service is running, with controls to stop and start it.
Installation name	A friendly name for this installation — what users see when choosing a server to connect to.
Server ID	The installation's unique identifier, with an action to issue a new one.
New certificate	Regenerates the service's TLS certificate. See Chapter 30 — clients that have already trusted the old certificate will need to renew.
Listening ports	The HTTP and HTTPS ports the service listens on — the same values entered when adding a server in Site Manager.
Data source	The database name and provider.

A **Change password** action protects the utility itself, so that the service cannot be reconfigured by anyone who simply has access to the machine.

28.3 The installers page

The MCU service runs a small HTTP server on its own listening port (§28.2) and serves a page at `/web/installers` — for a default installation, `http://<host>:9000/web/installers`. The initial database is created with installer images for all three applications, so this page is populated from the moment the service first runs.

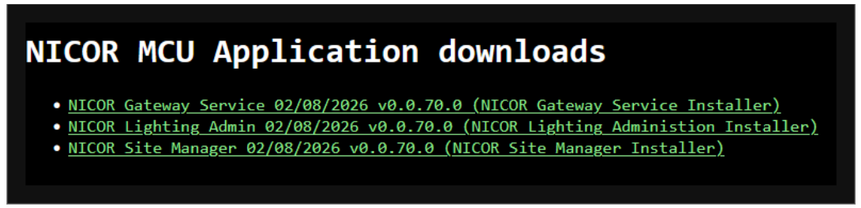


Figure 28.2. *The installers page. Every software image held in the database is listed, newest and older versions alike.*

The page lists *every* software image currently held in the database, not merely the newest of each — the same images the automatic update mechanism draws on. Uploading a new version adds it to the list rather than replacing what is there, so earlier builds remain downloadable alongside the current one.

> **WHY THE SERVICE ALONE CAN INSTALL EVERYTHING**
>
> Because the installer images live in the database and the service hands them out over HTTP, a whole installation can be built from the MCU service and its initial database. Install that one component, browse to its installers page, and pull the **Administration App** and **Site Manager** onto the machines that need them.
>
> The images are served over plain HTTP on the service's HTTP port, not its HTTPS port — convenient during first setup, when certificates may not yet be trusted, but worth knowing where a site restricts unencrypted traffic.

28.4 Creating the database

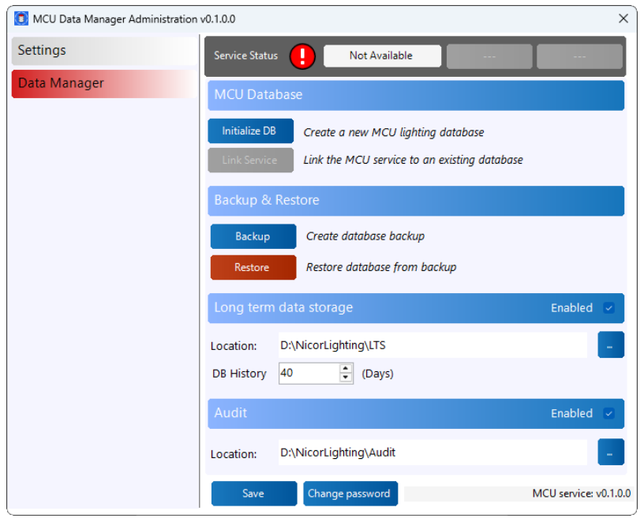


Figure 28.3. *The Data Manager page: database creation, backup and restore, long-term storage and audit.*

On a new installation, **Initialize DB** creates a clean database with the basic data required to start, and then automatically links the service to it — granting the database and system permissions needed along the way. This is the normal route, and it is a single action.

Link Service performs only the linking half and exists for a narrower purpose: where several incarnations of the lighting database already exist and a different one is to be selected, typically for engineering or testing. In ordinary operation it is not needed.

28.5 Backup and restore

Backup writes the database to a backup file. **Restore** reads one back, replacing the current database with its contents.

> **Restore requires the service to be stopped**
>
> The Restore action is **disabled while the service is running** and becomes available only once it has been stopped. This is a guard rather than a warning — the operation cannot be started against a live service. Restore is also **destructive and not undoable**: unlike changes made in the editor, there is no way back. Take a fresh backup before restoring an older one.

Two limits on scope are worth knowing:

- A backup contains the **database only** — not the long-term storage archive or the audit log, which live in their own folders on disk and must be backed up separately if they are wanted.
- Because historical energy data older than the retention window has already been moved out to long-term storage, a database backup contains only the *recent* portion of that history. Chapter 31 explains the split.

While the service is stopped for a restore, the installation behaves as described in Chapter 2: applications cannot log in,

running gateways continue on what they already hold, and lighting is unaffected.

28.6 Long-term storage and audit

The lower half of the page governs two on-disk stores, each with its own location and an option to enable it.

Long term data storage is the archive for historical energy data. The **DB history** setting alongside it — a number of days — is how long that data stays in the live database before being moved out to the archive. This applies to historical energy data alone; configuration is never aged out, and zone state history is not stored in the database at all. Chapter 31 sets out the full retention model.

Audit records activity against the system for later review, written to its own location. Chapter 32 covers using it.

> **Both are separate from the database backup**
>
> Because these stores live outside the database, they are not captured by the Backup action. A complete backup strategy covers three things: the database file produced here, the long-term storage folder, and the audit folder.

PART IV · FOR SYSTEM ADMINISTRATORS

29 The Gateway

The gateway is the service that stands between the MCU and the controllers on site. This chapter covers its administration utility — identity and network, operating options, email, notifications, and the buildings it serves.

29.1 What the gateway does

A gateway loads its assigned buildings from the DataManager, delivers their configuration down to the IMS controllers, reads live status back, and handles the notifications and BACnet exposure associated with them. An installation may run several gateways — though typically one per site.

It is the piece whose absence produces the "Live status unavailable" condition described in Chapter 2: with the gateway down, configuration remains readable and editable, but nothing live can be seen or driven, and changes wait until it returns.

29.2 Network and identity

The Network page establishes which gateway this service is and how it can be reached.

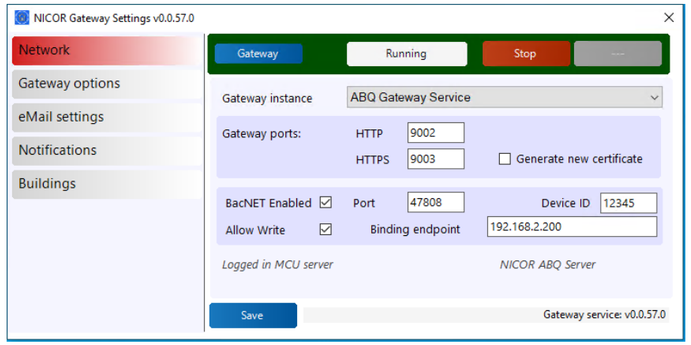


Figure 29.1. *The gateway's network page — instance, ports, certificate and BACnet.*

The **gateway instance** dropdown is the important one: it selects which of the registered gateway services (Chapter 26) this running service actually is. Registration creates the record; this is where a service claims it. The page also shows which MCU server the service is logged in to.

Alongside sit the service's **HTTP and HTTPS ports** and an option to **generate a new certificate**, which regenerates its TLS identity — see Chapter 30 for what that means for anything already trusting it.

The lower part of the page configures **BACnet**: whether it is enabled, the port and device identifier it presents, the network endpoint it binds to, and whether external systems are permitted to **write** as well as read. Appendix A covers the integration itself; the setting to weigh carefully here is write access, since it determines whether a building management system can drive the lighting or merely observe it.

29.3 Operating options

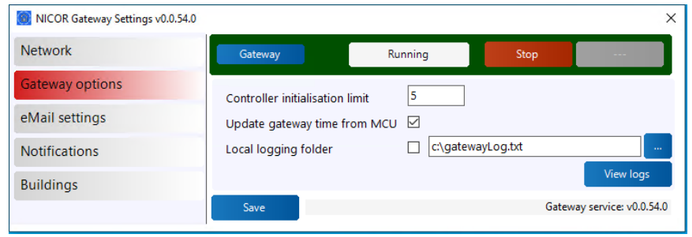


Figure 29.2. *Gateway options.*

Option	Purpose
Controller initialization limit	Caps how many controllers the gateway brings up at once. On a large site, initializing everything simultaneously is heavy going; this paces the work.
Update gateway time from MCU	Keeps the gateway's clock aligned with the MCU server — which matters because the gateway in turn keeps the controllers aligned, and they compute sunrise and sunset locally.
Local logging folder	Enables logging to disk at a chosen location, with an action to view the logs. Useful when investigating a problem; see Chapter 32.

29.4 Email

Before the gateway can send anything, it needs an outgoing mail server: host and port, the account and password to authenticate with, and the sender address messages will appear to come from. A **Verify** action tests the settings, which is worth using at setup rather than discovering a problem when the first notification fails to arrive.

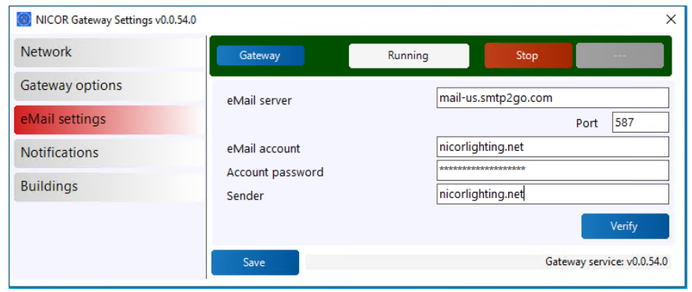


Figure 29.3. *Outgoing mail settings, with Verify to test them.*

29.5 Notifications

With mail configured, the gateway can send two notifications covering *all* the buildings it serves — a summary and a building log — each with its own recipients, and a set of day buttons choosing when they go out.

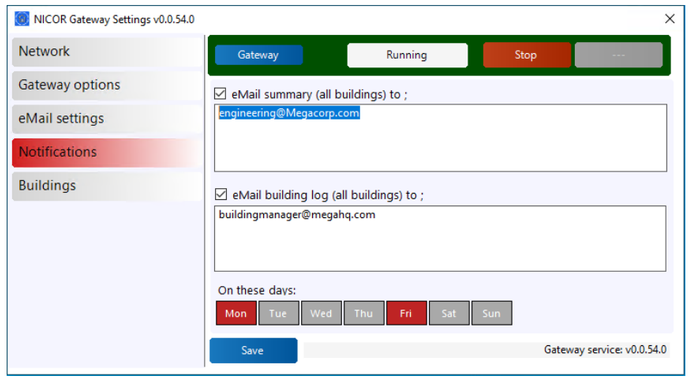


Figure 29.4. *All-buildings notifications.*

> **Two independent notification systems**
>
> These are separate from the per-building notifications configured in Chapter 13, and the two operate independently. A recipient can therefore be subscribed to both levels and receive two messages, or configured at neither and receive none while assuming they are covered. When someone reports duplicates or silence, check both.

29.6 Assigned buildings

The final page lists the buildings this gateway serves, with their identifiers. It is **read-only** — a building is assigned to a gateway from the building's own properties (Chapter 13), not from here.

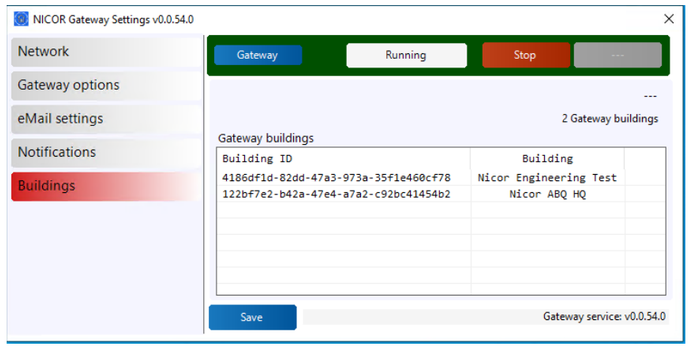


Figure 29.5. *The buildings this gateway serves — a read-only view.*

It is nonetheless useful for confirming that an assignment has taken effect: if a building has been pointed at this gateway but does not appear here, the two are not talking, and the connection is the thing to investigate rather than the building's configuration.

30 Security and Certificates

Connections between the MCU's parts are secured with TLS. This chapter explains the trust model — which is a little different from the one familiar from public websites — and what to do when a certificate changes.

30.1 How connections are secured

Traffic between the applications, the DataManager and the gateways is encrypted using TLS, so credentials and configuration are not exposed on the network. Each service holds its own **certificate**, generated for the machine it runs on, and presents it when something connects.

These certificates are **self-signed** rather than issued by a public certificate authority. That is a deliberate choice for a system installed on private networks: it means an installation needs no external dependency, no purchased certificates and no internet-facing name in order to be secure. It does, however, change how trust is established — which is the subject of the next section.

30.2 Trust on first use

A web browser trusts a public website because a certificate authority vouches for it. With self-signed certificates there is no such authority, so the MCU uses a different and well-established approach: **trust on first use**.

The first time an application connects to a server, it records that server's certificate. On every connection afterwards it checks that the certificate presented is the same one. If it matches, the connection proceeds silently. If it does not, something has changed — and the connection is refused until the change is accepted deliberately.

> **What this protects against**
>
> The model does not try to prove a server is who it claims to be on first contact. What it does is guarantee that the server you are talking to today is the *same one* you talked to before — so an impostor cannot quietly take its place. This is why servers are identified by host name rather than a bare address: the identity being pinned belongs to a named machine.

30.3 When a certificate changes

A certificate changes for one of a few reasons:

Cause	Notes
Deliberate regeneration	Requested on the DataManager settings page or the gateway network page — for example as part of a security policy, or after a suspected compromise.
Reinstallation or migration	A service rebuilt or moved to a new machine generates a new certificate for that machine.
Expiry and renewal	Certificates do not last indefinitely; a renewed one is a different certificate.

In each case, everything that had trusted the old certificate now sees an unfamiliar one and stops connecting. This is the system working as intended — but it does mean a planned change needs a plan for re-establishing trust afterwards.

30.4 Renewing trust

On the client side, **Renew Certificates** in the login window's settings menu re-establishes trust with the servers it connects to — discarding what was previously recorded and accepting what is presented now.

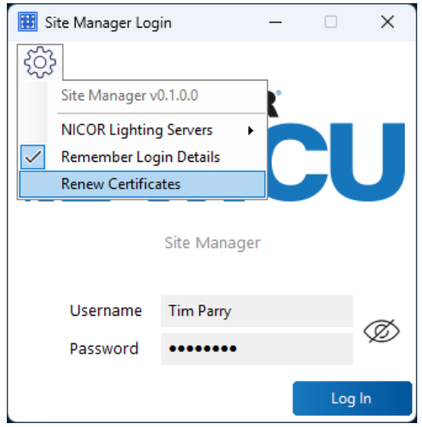


Figure 30.1. *Renew Certificates, in the login window's settings menu.*

> **Use it deliberately, not reflexively**
>
> Renewing trust accepts whatever certificate the server is currently presenting. That is exactly right after a change you made or expected — and exactly wrong as a reflex when a connection unexpectedly fails, because the refusal may be the system telling you something genuinely unexpected has happened. If a certificate has changed and you do not know why, find out why before renewing.

30.5 Practical guidance

- Regenerate certificates **during a maintenance window**, not casually — every client and gateway trusting that service will need trust re-established afterwards.
- Regenerating a **gateway's** certificate affects what connects to that gateway; regenerating the **DataManager's** affects every application and gateway on the installation. The second is much the larger event.
- Remember the resilience model: while trust is broken, controllers keep running the lighting. The urgency is about restoring *management*, not about the building going dark.
- Keep the administration utilities' own passwords in order too. TLS protects traffic; it does not stop someone with access to a server reconfiguring the service.

31 Data Retention

The MCU keeps three quite different kinds of data, in three different places, for three different lengths of time. Confusing them leads to unpleasant surprises — so this short chapter sets out what is kept where, and for how long.

31.1 The whole picture

Data	Held in	Kept for
Configuration clients, users, licenses, buildings, zones, devices, schedules, the catalogue	The database	**Permanently.** Never aged out or archived.
Historical energy data the basis of energy reporting	The database, then long-term storage	**Indefinitely,** but it moves: recent data sits in the database, older data is archived out to long-term storage, which has no limit.
Zone state history the per-zone timeline and light levels	The gateway	**24 hours,** on a rolling basis. Never written to the database at all.

31.2 Configuration is permanent

Everything that describes how the system is set up — the customer structure, user accounts, buildings and their areas, zones and their settings, devices, schedules and the device catalog — lives in the database and stays there. None of it is subject to any retention setting.

The retention controls in the DataManager utility do not touch it, so there is no risk of configuration quietly ageing away. Configuration is lost only if it is deleted deliberately or the database itself is lost.

31.3 Energy data moves rather than expires

Historical energy data is the one class subject to the **DB history** setting — a number of days, configured in the DataManager utility (Chapter 28). What that setting does is often misread, so it is worth stating plainly: it is not a deletion policy. When data passes the window, it is **moved out of the database and into long-term storage**, which has no size or age limit of its own.

The practical effect is a two-tier arrangement that users never see. A report covering last week is answered from the database; a report covering last year draws on the archive as well. Nothing is lost at the boundary and no user action is needed to cross it.

What the DB history setting is really for

It controls how large the live database is allowed to grow, not how much history you keep. A shorter window keeps the database lean and its backups small, at no cost to the reports — provided long-term storage is enabled. If it is *not* enabled, there is nowhere for aged data to go, and the window becomes exactly the limit of how much history you have.

31.4 Zone state history is transient

The per-zone timeline shown on a zone's History page — which state it was in, and the light level it measured — is different in kind from the other two. It is held **in the gateway**, not in the database, and covers a rolling **24 hours**. Older details are simply discarded.

This is deliberate: it exists to help commission and troubleshoot a space, where what matters is what happened this morning, not last quarter. Two consequences follow. It cannot be used for long-term analysis — the energy reporting in Chapter 10 is the tool for that. And it does not survive a gateway restart or outage, so if a zone has misbehaved and the history matters, look at it before restarting anything.

31.5 The audit log

Separately from all of the above, the audit log records activity against the system and is written to its own location on disk, configured alongside long-term storage. Because it lives outside the database, it is governed by neither the DB history setting nor the database backup. Chapter 32 covers using it.

31.6 What this means for backups

The three-way split has a direct consequence that is easy to get wrong. A database backup captures the configuration and whatever energy data currently sits inside the database — and nothing else. Specifically:

- Long-term storage is **not** included. Back up its folder separately, or older energy history is unprotected.
- The audit log is **not** included. Back up its folder separately if it is needed for compliance.
- Zone state history is not backed up by anything, because it is never stored — which is correct, and worth knowing so nobody goes looking for it.

A complete backup covers three things

The database file, the long-term storage folder, and the audit folder. Backing up only the first is a common and reasonable-looking mistake — it protects the system's configuration completely while leaving most of its history exposed.

PART IV · FOR SYSTEM ADMINISTRATORS

32 Monitoring and Diagnostics

When something is not behaving, the question is usually where in the chain the problem lies. This chapter covers the tools for answering that — the gateway report, the log files and the audit log — and suggests an order to work through.

32.1 The gateway report

The gateway report is the single most useful diagnostic in the system. It shows what the gateway and its controllers have actually been doing, as a running list of events with a status summary above.

It can be opened two ways: from a building's **Gateway** menu in Site Manager, showing that building; or from the Gateway Administration App, covering every building the gateway serves.

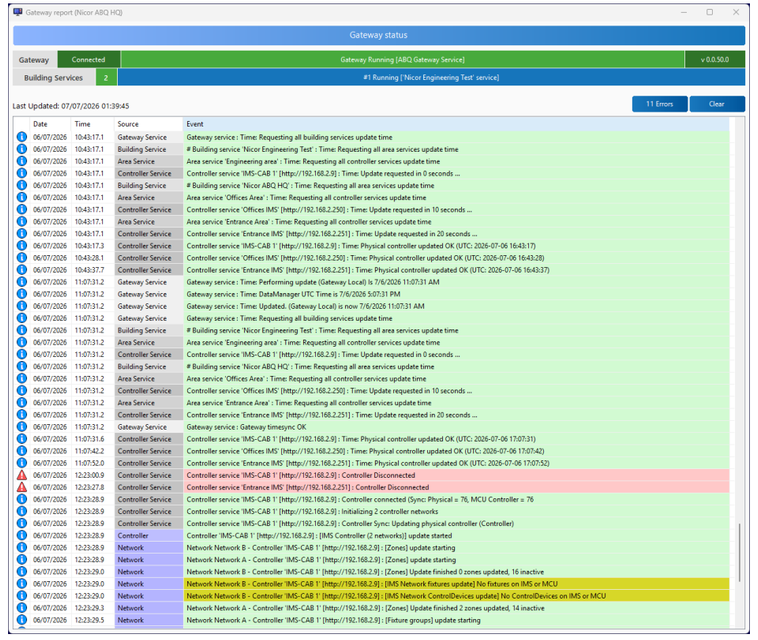


Figure 32.1. *The gateway report: status above, and the event list below with sources and color-coded severity.*

32.2 Reading it

The header confirms the basics — whether the gateway is connected, which service it is and its version, and how many building services are running. If something is wrong at this level, nothing below it will be healthy either.

The list beneath is where the details are. Each entry carries a timestamp, a **source** and a description, and the sources follow the system's own hierarchy:

Source	Reports on
Gateway service	The gateway itself — startup, time synchronization, and its own updates.

Source	Reports on
Building / Area service	Work at building and area level, including requesting updates for the objects beneath them.
Controller service	Communication with a particular IMS — connections, disconnections and synchronization.
Controller / Network	The details of what was updated on the hardware — zones, fixtures, input devices and groups.

Entries are color-coded by severity, and an **error count** in the header lets you jump to the problems rather than reading everything. A **Clear** action empties the list — which is genuinely useful before reproducing a fault, since whatever appears next is certain to be related.

32.3 What healthy looks like

Because the report is busy even when nothing is wrong, it helps to recognize the normal pattern. A routine synchronization reads roughly as: the gateway requests updates for its building services; each area service requests updates for its controllers; each controller service reports that the physical controller was updated successfully; and the controller and network entries then detail what was written — zones updated, fixtures and control devices checked, groups applied.

Seeing that sequence complete is confirmation that a change you made in the editor has actually reached the hardware. Its *absence* after a save is itself informative: it suggests the change has not been delivered, which points at the gateway connection or the controller's network mode rather than at the configuration.

> **Not every warning is a fault**
>
> Some entries are simply the system reporting a benign condition — a network with nothing on it, for example, or an object that needed no change. Reading a few reports on a healthy installation is the quickest way to learn which entries are routine, so that genuine problems stand out later.

32.4 Log files

For anything the report does not settle, the gateway can write a log to disk. Logging is enabled and its location chosen in the Gateway Administration App (Chapter 29), which also provides an action to view what has been written.

Logs are the right tool for intermittent problems and for anything needing to be sent on for support: the report shows the current model, whereas a log persists and can be examined after the fact.

32.5 The audit log

The audit log answers a different question: not "what is the system doing?" but "who did what?" It records activity against the system, and is enabled with its own location in the DataManager utility (Chapter 28).

It is the place to look when a configuration change needs to be accounted for — who altered a schedule, or when a building's settings last changed. As noted in Chapter 31, it lives outside the database and is therefore not covered by a database backup.

32.6 A diagnostic order

Most reported problems resolve quickly if the chain is checked from the top down rather than starting where the symptom appeared:

1. Is it **licensing or permissions**? If nobody can do something, suspect licensing (Chapter 25); if one person cannot, suspect permissions (Chapter 23).
2. Is the **gateway connected**? "Live status unavailable" across a building answers this immediately — and remember the lighting itself will be running regardless.
3. Is the **controller connected and in the right mode**? A controller left in Read-only after commissioning will never receive changes, which presents as configuration that mysteriously does not take effect.
4. Did the **change reach the hardware**? Check the gateway report for the synchronization sequence following your save.
5. Is the **device model correct**? Addressing mismatches present as the wrong light responding, or none at all — see Chapter 18.
6. Is the **zone behaving as configured**? Use the quick-test controls (Chapter 9) to exercise it directly, and the zone's history to see what it has been doing.

> **Start with what is cheap to check**
>
> The first three steps take seconds and account for a large share of reported faults — particularly "the configuration will not apply", which is very often a controller still in Read-only mode. Working downwards also has the advantage of ruling out whole layers at a time, rather than investigating a zone whose controller was never connected in the first place.

PART V · APPENDICES

Appendix A

BACnet Integration

The MCU can present its zones to a building management system over BACnet/IP, both for monitoring and — optionally — for control. This appendix is written for the integrator on the other side of that link: what is exposed, how it is named, what the values mean, and what happens when the BMS writes. The exposure is deliberately narrow, and the write semantics are stronger than they first appear.

A.1 What is exposed

One thing only: **zones**. Every zone belonging to a building served by the gateway is presented as a BACnet object, and nothing else is. There are no objects for fixtures, input devices, networks, controllers, areas or buildings, and no energy or diagnostic data is available over BACnet.

What each zone object carries is its **state** — occupied, unoccupied and so on — not its light level. A custom property for zone light level existed during development and was removed; a BMS can therefore learn whether a space is in use, and can command it, but cannot read or set a dimming level.

Exposure is license-gated per client. Zones belonging to a client whose licensing does not include BACnet are not published at all, rather than being published and refused (Chapter 25).

> **THE NARROWNESS IS THE DESIGN**
>
> A BMS integration that exposed every fixture would produce thousands of points, most of which no BMS operator would ever use, and would invite exactly the fixture-by-fixture control that the zone model exists to avoid.
>
> Zones are the level at which the MCU makes decisions, so they are the level at which it is worth another system participating. If a BMS needs to influence lighting in a space, the zone is the correct object to reach for.

A.2 Enabling and connecting

BACnet is configured per gateway service, on the Network tab of the Gateway Administration App (Chapter 29).

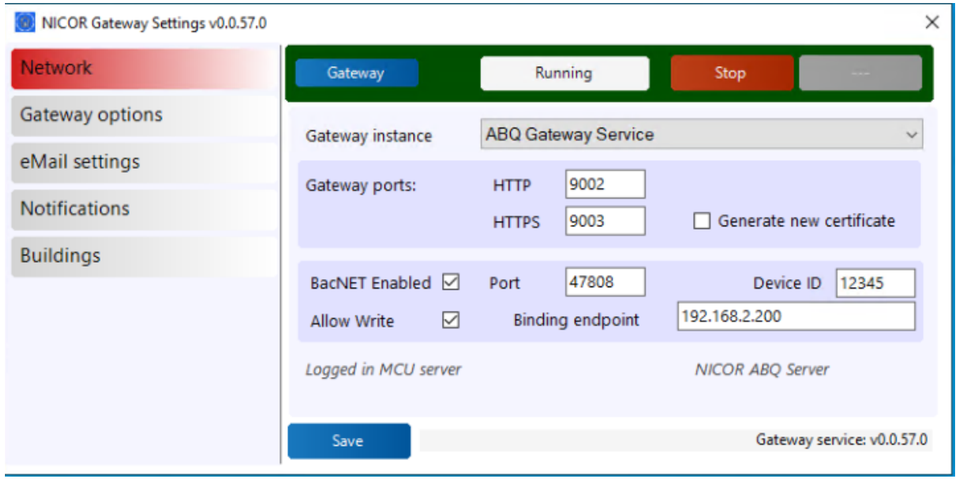


Figure A.1. *The Gateway Administration App, Network tab. The BACnet settings occupy the lower half of the panel; the HTTP and HTTPS ports above belong to the gateway itself.*

Setting	Meaning
BacNET Enabled	Whether the gateway publishes a BACnet device at all.
Port	The UDP port. 47808 is the BACnet/IP default.
Device ID	The device instance number by which the gateway is identified on the BACnet internetwork. It must be unique across the internetwork — coordinate it with whoever administers the BMS.
Binding endpoint	The local IP address the service binds to. On a machine with more than one interface this decides which network the device appears on.
Allow Write	Whether the BMS may command zones as well as read them. See §A.6 before enabling it.

Two further identifiers appear on the device object and are not configurable. The **Vendor Identifier** is **1625**, assigned to NICOR Lighting and fixed in the gateway — it is what confirms in a discovery listing that a device really is an MCU gateway rather than something else answering on the same port. The device instance number, by contrast, is the Device ID above and is yours to set.

A.3 The object list

The gateway presents a single BACnet device. Beneath it, each zone appears as an object of type OBJECT_MULTI_STATE_VALUE in a flat list — there is no hierarchy in the BACnet view, and no structured view or grouping object. The hierarchy is carried in the object names instead.

Each object is named by its full path through the MCU's hierarchy:

```
Building/Area/Controller/Network/Zone name
```

The separator is a forward slash with no surrounding spaces, which matters to anyone matching names programmatically.

A zone with no name of its own appears as "Untitled", which is worth avoiding: on a site of any size the BMS point list is the only place many people will ever see these names, and several identically-named entries are of no use to anyone.

BIND TO THE OBJECT NAME, NOT THE INSTANCE NUMBER

Zone object instance numbers are assigned in the order the zones are added, starting at 1 — the device object itself occupies 0. They are positional, and they are **not guaranteed to be stable**. A zone added, removed or reordered can shift the numbering of others.

An integration that binds points to instance numbers may therefore quietly point at the wrong rooms after a configuration change, with no error anywhere to indicate it. Binding to the object name costs nothing at commissioning time and removes the whole class of problem.

Alongside the standard properties, each zone object carries a custom property holding the zone number — the DALI group number, 0 to 15, described in Appendix C §C.3. It is available to clients that can read non-standard property identifiers and can be ignored by those that cannot.

A.4 Zone state — the present value

The object's Present_Value is the zone's current state, as a multi-state value. State_Text carries the names, so a well-behaved client will display them without any configuration. Number_Of_States is 5, but only the first four are states a zone can be in — the fifth exists solely as a write value and is covered in §A.6.

Value	State	Meaning
1	Unoccupied	Nothing detected; the zone is at its unoccupied level.
2	Preliminary	Something detected, not yet committed to full occupancy.
3	Occupied	The space is in use.
4	Timeout	Counting down towards unoccupied.
5	Release to IMS	Write only. Releases a BACnet lock and returns the zone to automatic control — see §A.6. Never returned by a read.

These four are the state machine described in Chapter 4, unchanged — a BMS watching this value is watching the same thing the floorplan shows. A read will only ever return 1 to 4; the fifth value is a command rather than a condition and is never reported back.

A.5 Status flags

Three internal conditions drive three standard properties, which are also summarized into the four-bit Status_Flags string most clients display. Each bit is set when its condition is bad, in the order below, so a healthy, enabled, reachable zone reads 0000.

Property	Status_Flags bit	Set when
Event_State	IN_ALARM (first)	The gateway holds one or more warnings for the zone (Chapter 5 §5.1). Reported as a fault event state; normal otherwise.
Reliability	FAULT (second)	The gateway is not connected to the zone's controller, or is connected but has not yet successfully queried the zone's state. Reported as a communication failure.
—	OVERRIDDEN (third)	Never. This bit has no corresponding property and is always clear.
Out_Of_Service	OUT_OF_SERVICE (last)	The zone is disabled in the MCU.

IN_ALARM AND FAULT ARE NOT THE SAME KIND OF NEWS

They answer different questions, and a BMS alarm strategy that treats them alike will be wrong in both directions. **IN_ALARM** means the MCU is watching the zone and has found something to report — a warning it understands. **FAULT** means the MCU cannot see the zone reliably, so Present_Value should not be trusted.

The second is usually not a lighting problem at all. Because the IMS runs zone logic locally and autonomously, a zone reporting communication failure is very probably still detecting occupancy and driving its fixtures exactly as configured. What has been lost is visibility, not control.

FAULT also covers the interval between the gateway connecting and its first successful state query, so it can appear briefly during startup and clear on its own. A short delay before alarming on it avoids a burst of alarms every time a gateway service restarts.

SUPPRESSED WARNINGS NEVER RAISE IN_ALARM

Where warnings have been suppressed for a zone, the condition behind IN_ALARM is treated as good *unconditionally* — the bit will never be set for that zone, however many warnings it would otherwise carry.

This matters before designing anything around IN_ALARM. A zone that never alarms may be genuinely healthy, or may simply be one whose warnings were suppressed at some point and never restored, and the BACnet view gives no way to tell the two apart. That check has to be made in the MCU.

OUT_OF_SERVICE is different in kind from the other two. It means the zone has been disabled in the MCU — a configuration state somebody chose, not a failure. Standard BACnet uses the flag to mean an object decoupled from its physical I/O, which is close enough in spirit, but an integrator expecting the literal meaning should know what it is actually reporting here.

A.6 Writing to a zone

If Allow Write is enabled, a BMS may write 1, 2, 3 or 4 to a zone's Present_Value. This does not nudge the zone or express a preference. It takes the zone over.

A WRITE IS A LOCK, NOT A REQUEST

Writing a state locks the zone into that state. Every other means of controlling it becomes inactive — occupancy no longer moves it, timeouts no longer apply, inputs no longer act on it — on the assumption that a BACnet client has assumed control. This is deliberate: it stops the BMS and the IMS fighting each other, each undoing the other's decisions.

The zone stays locked until the client writes **5**, which appears in the state list as **Release to IMS**. That returns the zone to automatic control. Nothing else clears the lock.

A locked zone is visible on the plan: Site Manager overlays a flashing padlock on the zone icon while an external client holds it (Appendix B §B.2). That is the only outward sign, and it is worth knowing on sight — a locked zone otherwise looks entirely normal, showing whatever state has been commanded.

Two consequences follow, and both matter more than they look.

- **A client that disconnects without writing 5 leaves the zone locked indefinitely.** Occupancy detection in that room simply stops working, permanently, with no fault reported anywhere — because as far as the system is concerned nothing has gone wrong. Any integration that takes control should release it explicitly, including on shutdown and on error paths.
- **The release value is deliberately inside the declared range.** Number_Of_States is 5 precisely so that 5 is a legal

write, and a client that validates writes against the declared range will send it without complaint. Writing 5 is the only way to hand control back.

Where a BMS only needs to observe occupancy — for ventilation, heating or utilization reporting, which is the common case — leave Allow Write clear. Monitoring needs no write access and a read-only integration cannot strand a zone.

A.7 Change of value

COV subscription is supported, and is the right way to follow zone state. Zone state changes are infrequent and event-driven by nature, so a subscription delivers changes as they happen without the polling traffic that following several hundred zones would otherwise generate.

Notifications are not limited to Present_Value. The status flags described in §A.5 are re-evaluated continuously and a change of value is issued whenever the bit string actually changes, so a subscribed client learns that a zone has become unreadable at the moment it happens rather than on its next poll. A client that subscribes for state but polls for health is doing more work than it needs to.

A.8 Before commissioning an integration

1. Confirm the client's licensing includes BACnet, or no zones will be published.
2. Agree the Device ID with whoever administers the BACnet internetwork, and check the binding endpoint is on the network the BMS can reach.
3. Decide whether write access is genuinely needed. If it is not, leave Allow Write clear.
4. Name every zone. Unnamed zones appear as "Untitled" in the point list.
5. Bind points by object name rather than instance number.
6. If writing, make sure the integration releases every zone it locks by writing **5** — on shutdown and on failure as well as in normal operation.
7. Set the BMS alarm strategy to distinguish IN_ALARM from FAULT. The first means the MCU has found something to report; the second means it cannot see the zone reliably, which is usually a visibility problem rather than a lighting one.
8. Allow a short delay before alarming on FAULT, so the interval between a gateway restarting and its first successful state query does not produce a burst of alarms.
9. Check whether warnings have been suppressed on any zone. A suppressed zone can never raise IN_ALARM, and nothing in the BACnet view reveals that — it has to be checked in the MCU.

> *The zone is the only object the MCU exposes, and a write to it is a handover rather than a hint. Both facts follow from the same idea: BACnet participates in the MCU's decisions at the level where those decisions are made and does not compete with them.*

PART V · APPENDICES

Appendix B

Icon Legend and Canvas Conventions

A floorplan says a great deal in very little space. Every object on the canvas carries a symbol whose shape says what kind of thing it is and whose color says what state it is in, and the same symbols reappear in properties windows, the zone-state matrix and the input tree. This appendix is the key to them. It closes with the one thing about the color scheme genuinely worth learning: red does not mean the same thing everywhere.

B.1 Shape and color

Two rules cover almost everything on the canvas. Shape identifies the kind of object — a bracket is a zone, a chip is a controller, a bulb or dial or switch is a fixture, a diamond is an input device. Color identifies the state that object is currently in. Shape is worth learning once; color is what you read continuously.

There is one deliberate exception. The small zone-state symbols in §B.3 are pictograms rather than colored shapes, because in the places where they appear — the zone-state matrix, quick-test popups, History badges — color is already being used to carry something else. Where color is busy, the symbols carry the meaning instead.

B.2 Zone markers

A zone is drawn on the plan as four corner brackets around the area it covers. The color of those brackets is the zone's current state, using the same four states as the state machine in Chapter 4.

	Icon	What it means
	**Unoccupied**	Grey. Nothing has been detected; the zone is at its unoccupied level.
	**Preliminary**	Amber. Something has been detected but the zone has not yet committed to full occupancy.
	**Occupied**	Yellow. The zone is occupied and lit accordingly.
	**Timeout**	Blue. No movement has been reported recently; the zone is counting down towards unoccupied.
	**Status unknown**	Red. The MCU cannot say what state this zone is in — see below.

RED ON A ZONE MARKER IS NOT A FAULT IN THE ZONE

Red means the MCU does not know what the zone is doing, most often because the parent controller is not connected. It does *not* mean the zone has stopped working.

The distinction matters because the IMS runs zone logic locally and autonomously (Chapter 2). A zone whose marker is red is almost certainly still detecting occupancy, still timing out and still driving its fixtures exactly as configured — the lights in that room are fine. What has been lost is visibility, not control. Red is "I cannot see", not "it is broken".

Overlays

A zone marker can carry an overlay in addition to its state color. At present there is one.

	Icon	What it means
	**Locked**	A flashing padlock over the zone icon. The zone's state is being held externally by a BACnet client and its own logic is inactive. Appendix A §A.6.

THE PADLOCK IS THE ONLY OUTWARD SIGN

A locked zone looks entirely normal apart from this overlay. Its marker still shows a state color — whatever state the external client has commanded — and nothing anywhere reports a fault, because a lock is not a fault. Occupancy, inputs and timeouts are simply inactive while it is held.

So the padlock is worth knowing on sight. It is the difference between a room whose lighting has stopped responding for a reason somebody chose, and one that needs investigating.

B.3 Zone-state symbols

The same four states appear as small monochrome pictograms wherever they need to be identified without color — in the zone-state matrix on a zone's Settings tab, in quick-test popups, and as badges on the History tab.

	Icon	What it means
	**Unoccupied**	Four corner marks turned inward — an empty space.
	**Preliminary**	A partial figure — something is there, but not yet confirmed.
	**Occupied**	A standing figure — the space is in use.
	**Timeout**	A clock — the state defined by time passing rather than by anything detected.

B.4 Controller icons

A controller appears on the plan as a chip. Its color and the symbol inside it report both the connection and how the two configuration models compare. Chapter 19 §19.3 covers what each condition means for synchronization.

	Icon	What it means
	**Connected**	Green. Connected and in step. The ordinary state.
	**Status unknown**	Green with a question mark. Connected, but no status has been received from the controller yet.
	**Busy**	Blue. Performing initial setup or synchronization. Direct commands are unavailable until it finishes.
	**Disconnected**	Red cross. The Gateway cannot reach the controller at all.
	**IMS newer**	Red, arrow down. The controller holds configuration changes the MCU does not.
	**MCU newer**	Red, arrow up. The MCU holds changes ready to push down to the controller.
	**Physical devices changed**	Red. The two device and address models do not agree — see below.

THREE RED STATES, TWO QUITE DIFFERENT PROBLEMS

The two arrows describe a **revision difference**. The two sides agree about what exists and disagree only about how recently it was configured, and the synchronization machinery resolves that on its own once permitted to (Chapter 19).

Physical devices changed is a different kind of trouble. The two sides do not agree about *what exists* — devices have been added, removed or re-addressed. No amount of synchronizing will fix it, because there is nothing coherent to synchronize. That needs reconciling instead (Chapter 18).

Also worth separating: Status unknown means nothing has been heard yet, while Disconnected means contact was attempted and failed. The first often resolves itself within moments of a controller coming up; the second will not.

B.5 Fixture icons

A fixture's symbol shows how it is driven — its assigned role — rather than what kind of lamp it holds.

	Icon	What it means
	**DALI / standard**	A lit bulb. Ordinary DALI control gear, dimmable across the full range.
	**0-10V**	A dial. A converter driving non-DALI gear over an analogue control voltage.
	**Relay**	A switch. Switched output only — on or off, with nothing in between.
	**Unknown status**	A bulb with a question mark. The MCU cannot report what this fixture is doing.

SPOTTING RELAYS ON A PLAN EXPLAINS A LOT

Because the symbol shows the assigned role, a plan tells you at a glance which fixtures are relay-driven without opening anything. That is worth knowing, because relay fixtures behave differently from everything around them: they sit *outside* the zone's DALI group and follow a per-state on/off matrix rather than the zone's level (Chapter 16, and Appendix C §C.4).

When somebody reports that part of a zone refuses to dim, the switch glyphs on the plan usually answer the question before any dialog is opened.

B.6 Input devices and their instances

An input device is drawn as a diamond. On its own the diamond says only that a control device is present; what that device actually does is shown by the instance badges laid over it.

	Icon	What it means
	**Input device**	The base marker. Instance badges are overlaid on it.

Each badge names a kind of instance — button, motion sensor, light sensor — and distinguishes a device carrying a single instance of that kind from one carrying several. Appendix C §C.7 explains why one physical device commonly presents several instances at once.

THE SAME TWO COLORS, TWO DIFFERENT MEANINGS

On the floorplan, green and red on an instance badge mean **single** and **multiple**. On the instance grid of an input device's Details tab (Chapter 17 §17.6) the same two colors mean **used** and **unused**.

These are separate conventions serving separate purposes, and neither is wrong — but a red badge on a plan is not the same news as a red instance in a properties window. The first is a count; the second is a status. Read the color against the place it appears.

	Icon	What it means
	**Button — single**	One button instance.
	**Button — multiple**	More than one button instance on the same device, as on a multi-key wall plate.
	**Motion sensor — single**	One occupancy instance.
	**Motion sensor — multiple**	More than one occupancy instance.
	**Light sensor — single**	One light-sensor instance.
	**Light sensor — multiple**	More than one light-sensor instance.

B.7 Reading color

Across the canvas as a whole the palette is broadly consistent, and knowing the pattern lets you scan a plan quickly.

Colour	Generally means
Grey	Inactive or unoccupied. Nothing is happening, and nothing is wrong.
Amber / yellow	Occupancy — preliminary, then full.
Blue	A transitional condition: a zone timing out, or a controller busy.
Green	Healthy, or singular.
Red	Attention — but of several different kinds. See below.

RED IS DOING FOUR DIFFERENT JOBS

On a **zone marker** it means the state is unknown, usually because the controller is unreachable. The zone is probably running normally.

On a **controller** it means one specific condition — disconnected, a revision difference, or a device mismatch — and the symbol inside says which.

On a **fixture** it marks unknown status, again a visibility problem rather than a failure.

On an **instance badge** it simply means *more than one*, and is not a problem at all.

Check what kind of object you are looking at before reacting to the color. The only one of these four that reliably needs action is a controller, and even then the symbol decides whether the answer is in Chapter 18 or Chapter 19.

Appendix C

DALI-2 Concepts and Glossary

The MCU is a management layer over a protocol it did not invent. Almost every limit in this manual — sixteen zones, sixty-four fixtures, sixteen scenes, the existence of a timeout at all — comes from DALI rather than from a design decision at NICOR. Chapter 5 §5.3 gives the vocabulary needed to read the manual; this appendix supplies the reasoning behind it, explaining why each of those constraints exists and what follows from it. It closes with a glossary of every term the manual uses. Reference material: read it once, or come back to it when a limit seems arbitrary.

C.1 What DALI is, and what it is not

DALI — Digital Addressable Lighting Interface — is a control protocol for lighting, defined by the international standard IEC 62386. It carries instructions and replies. It does not carry the power that lights the lamps: mains reaches each luminaire by its own route, and the DALI pair runs alongside it. A DALI command tells a driver *what to do*; the driver then does it with the power it already has.

The word that matters in the name is **addressable**. Older control methods — phase-cut dimming, 0-10V, switched circuits — act on a wire, so the granularity of control is the granularity of the wiring. Rewiring is the only way to change which lights respond together. DALI addresses devices individually over a shared pair, so grouping is a property of the configuration rather than of the cable. A fixture can be moved from one zone to another without an electrician.

The second thing that matters is that DALI is **bidirectional**. Devices answer questions. A driver can be asked its current level, whether its lamp has failed, what its minimum output is; a sensor can announce that it has seen movement. This is what makes the reconciliation and diagnostic behavior described in Chapters 18 and 32 possible at all — the system can compare what it believes against what is actually on the wire.

THE SCALE DALI WAS DESIGNED FOR

A DALI network is deliberately small: one bus, sixty-four devices of each kind, and up to 300m (985ft). It is a *room-to-floor* technology, not a building network. Everything above that scale — many networks, many controllers, many buildings, scheduling, permissions, history — is the job of the layer above. That layer is the MCU.

C.2 The bus: wiring, power and speed

A DALI network is a single pair of wires, and every device on it connects to that same pair. The pair is polarity-free for most equipment, needs no termination and no screening, and may be run in the same multicore cable as the mains supply because it is rated to the same insulation standard. Topology is free: line, star, tree, spurs off spurs — anything except a ring. In practice this means the bus can follow the lighting circuit rather than being planned separately.

The bus carries its own low-voltage supply, provided by a bus power supply that is limited in current — 250 mA is the usual ceiling. Every connected device draws a small quiescent current from it, so the device count is bounded by power as well as by addressing. Cable length is bounded in turn by voltage drop, giving the familiar figure of a few hundred meters for sensible cable sizes.

Signaling is slow by any modern standard: 1200 bit/s, Manchester encoded. A command frame is 19 bits, so a single instruction occupies roughly sixteen milliseconds of the bus, and a query plus its reply and the mandatory settling time

takes appreciably longer. This is not a defect — the slow, heavily encoded signaling is what lets the bus survive being run next to mains in a metal conduits for three hundred meters — but it has a consequence that shapes the whole system.

> **WHY THE SYSTEM COMMANDS GROUPS, NOT DEVICES**
>
> Setting sixty-four fixtures individually means sixty-four frames, which is over a second of bus time, and the fixtures visibly step rather than move together. Setting them by group is one frame. This is why zones map onto DALI groups rather than being maintained as lists in software, and why scene recall (§C.6) is the mechanism of choice when several different levels must be reached at the same instant.
>
> It is also why polling every device for status is expensive, and why the MCU reads status on a cycle rather than continuously. Chapter 32 covers what that means for how quickly a fault becomes visible.

C.3 Addressing: short addresses, groups and broadcast

Short addresses and groups are introduced in Chapter 5 §5.3. They are worth setting out fully here, because between them they account for most of the numbers that bound an MCU network. DALI offers three ways to aim a command.

1. **Short address.** A number from 0 to 63 held by the device, assigned during commissioning. Sixty-four addresses is the hard limit on how much control gear one bus can hold. DALI-2 gives control devices an address space of their own, also 0 to 63, so a bus can carry sixty-four drivers and sixty-four sensors without collision.
2. **Group address.** A number from 0 to 15. Any device can be a member of any number of groups, and a command to a group is acted on simultaneously by all its members. Sixteen is the hard limit on groups per bus.
3. **Broadcast.** Every device on the bus responds. Broadcast works before any commissioning has happened, which makes it the tool of choice for proving a bus is alive.

> **WHERE THE MCU'S NUMBERS COME FROM**
>
> The per-network limits quoted throughout this manual — **16 zones, 64 fixtures, 64 input devices, 16 scenes** — are not NICOR limits. They are the DALI address spaces, exposed directly. Note that the sixty-four is *per kind*: a network holding sixty-four fixtures can still hold sixty-four input devices, because DALI-2 addresses control gear and control devices in separate spaces.
>
> In particular, a zone's number is not a label. Zone 5 *is* DALI group 5, and its fixtures are group 5's members. This is why zone numbers cannot be duplicated within a network, why moving a fixture between zones is a group-membership change on the device itself, and why a fixture belongs to exactly one zone while an input device may serve several. See Chapter 4.

C.4 Control gear and control devices

DALI divides the world in two. Control gear makes light: LED drivers, fluorescent ballasts, relay modules, converters that translate DALI into an analogue 0-10V signal for a non-DALI driver. Control devices ask for light: occupancy sensors, light sensors, push-button units. The MCU calls the first fixtures and the second input devices, and the distinction runs deep enough that the two are configured on entirely different screens (Chapter 17).

The original standard defined control gear thoroughly and control devices barely at all, so DALI-1 sensors were manufacturer-specific in practice — one vendor's sensor would not drive another's system. **DALI-2, published from 2014 onward, standardized control devices** and introduced certification and mandatory identity data. This is the single most important difference for anyone commissioning a NICOR system: an MCU network expects DALI-2 behavior from its input devices, and a DALI-1 sensor cannot simply be substituted.

Control gear declares a device type, identifying the extended command set it understands. Three of them map directly onto

the assigned role chosen for a fixture:

Device type	IEC 62386 part	What it is
DT6	Part 207	An LED module driver. The ordinary case, and the DALI assigned role.
DT7	Part 208	A switching device — a relay. On and off only, no intermediate levels. The Relay assigned role.
DT5	Part 206	Conversion of DALI into a DC control voltage, for driving 0-10V or 1-10V gear. The 0-10V assigned role.
DT8	Part 209	Color control — tuneable white and full color.
DT1	Part 202	Self-contained emergency lighting, with its own test and duration regime.

The Relay case is worth dwelling on, because it explains a behavior that otherwise looks like an inconsistency. A DT7 device has no dimming curve to sit on, so it cannot follow a zone's level. It therefore sits *outside* the zone's DALI group and is driven by a separate per-state on/off matrix applied across the whole zone — see Chapter 16.

C.5 Levels, the dimming curve and fades

DALI expresses light output as an arc power level from 0 to 254. Level 0 is off. Level 254 is maximum. The value 255 is reserved as MASK and means "no change".

The scale is **logarithmic, not linear**. It is designed so that equal steps look like equal changes to the human eye, which perceives brightness roughly logarithmically. Levels 1 to 254 are mapped across a range from about 0.1 % to 100 % of the device's rated output. The practical consequence surprises people the first time they meet it: the middle of the scale is nowhere near half the light.

Arc power level	Approximate share of rated light output
254	100 %
229	about 50 %
204	about 25 %
170	10.09 % — the figure the interface shows for this level
127	about 3 %
85	about 1 %
43	about 0.3 %
1	about 0.1 %
0	Off

READING THE TWO NUMBERS TOGETHER

Where the MCU shows a level it shows both figures, the raw value in square brackets followed by the percentage — `[170] 10.09%`. **The percentage is actual light output.** The raw value is the point on the logarithmic curve that produces it, and the two deliberately do not track one another: fifty per cent of light output sits at around level 229, not at the midpoint of 127.

The raw value is the one that surfaces in device settings, in diagnostics, and in anything read back from the bus. Read it as a position on a curve, never as a percentage in its own right — a fixture reporting level 127 is at roughly three per cent output, not half. This is also why estimated energy figures are derived from load-type power curves rather than from the level directly; see Chapter 10.

Levels a device holds for itself

Four settings live in the device and apply without anyone commanding them. They matter most precisely when the control system is not there.

- **MIN LEVEL and MAX LEVEL.** Per-device limits, surfaced by the MCU as **Minimum** and **Maximum** on a fixture's settings tab. A driver told to go below its MIN LEVEL settles at MIN LEVEL rather than switching off — only an explicit level 0 turns it off. Setting a zone below a fixture's minimum therefore produces a floor, not darkness.
- **POWER ON LEVEL.** What the device does when its mains supply returns, before any controller has spoken to it. Left at the factory value this is usually full output, which is why a circuit breaker being reset lights a building to full brightness for a moment.
- **SYSTEM FAILURE LEVEL.** What the device does when the bus itself fails — a cut cable, a dead bus power supply. Deliberately not "off", so that a wiring fault leaves people with light rather than in the dark. The MCU shows it as **System Fail**, and it is subject to the same inheritance convention as any other fixture setting (Chapter 5 §5.2).

These three are the last line of defense in the layered model this manual keeps returning to. The IMS runs the site whether or not the gateway is reachable; these device-held levels run the luminaire whether or not the *IMS* is reachable.

Fades

A commanded level is reached over a fade time rather than instantly. Fade time is chosen from sixteen fixed values rather than being set freely: the scale runs from about 0.7 seconds through 1, 1.4, 2, 2.8 seconds and so on, roughly doubling every two steps, up to about 90 seconds at the top. DALI-2 adds an extended fade time for the very short fades the original scale could not express. A separate fade rate governs the fade-up and fade-down commands that move without a stated destination — the behavior behind a held dimming button. The MCU shows fade time in seconds and fade rate in steps per second, the latter abbreviated "S/Sec"; both appear on a fixture's settings tab alongside the level limits.

C.6 Scenes

Chapter 5 §5.3 records that a scene is one of sixteen stored levels held *per device*. That detail is the whole point of scenes, and is worth drawing out. A scene is not a list kept by the controller; it is a level written into each device against a number from 0 to 15. Recalling it is a single command — "go to scene 4" — and every device already knows its own level for scene 4 and moves there immediately.

This is why the Scenes light method exists alongside Fixed power and Light levels (Chapter 16). If a zone needs its fixtures at *different* levels — brighter at the back of a room, dimmer near the window — a scene reaches that arrangement in a single frame, with every fixture moving together. Achieving the same thing by addressing each fixture in turn would take a second or more and would visibly ripple across the space.

The cost is that the arrangement lives in sixty-four separate devices. Changing a scene means writing to every fixture in it, and a replaced fixture arrives from its box knowing nothing about the scenes it is expected to join. This is a routine cause of one luminaire behaving oddly after a swap, and Chapter 18 covers restoring it.

MASK, OR "NO CHANGE"

A device whose stored value for a scene is MASK does not participate: recalling that scene leaves it exactly where it was. This is how a scene can address part of a zone and deliberately ignore the rest. The MCU renders it on a fixture's scene slider as **No Change** rather than as a level — so a scene reading "No Change" is not an unconfigured scene, it is a deliberate instruction to stay put.

C.7 Instances, events and the reason timeouts exist

Chapter 5 §5.3 introduces instances: a control device is a container, and the parts that actually sense are instances within it — the separate buttons of a wall plate, the occupancy and light-level functions of a multi-sensor. Two details matter beyond that. Each instance is numbered from zero within its device and declares an instance type saying what it does, so the system knows a sensor's capabilities without being told. And because one physical device can serve several purposes in several places, input devices are shared between zones rather than owned by one, which is the distinction that governs the copy, move and delete behavior in Chapter 17.

Instance type	What it reports
301	Push buttons — press, release, short press, long press, and the repetitions between them.
302	Absolute input devices — sliders and dials, reporting a position.
303	Occupancy sensors — movement detected.
304	Light sensors — measured illuminance.

Control devices are event-driven. They are not polled; they speak when something happens. And this is where the most consequential fact in the whole protocol appears.

OCCUPANCY SENSORS CANNOT REPORT ABSENCE

An occupancy instance reports one thing: *movement has been detected*. There is no message for "the room is empty", because emptiness is not an event — it is the continued absence of one, and a sensor cannot distinguish an empty room from a still one, or from a failed sensor, or from a cut cable.

Every timeout in the MCU exists to answer this. Because absence can never be announced, it must be **inferred from the passage of time** since the last announcement of presence. That inference is exactly what the zone state machine's timeout values encode, and it is why a zone has a Preliminary and a Timeout state rather than simply "on" and "off". See Chapter 4.

Part 303 gives an occupancy instance three timers of its own, below the MCU's state machine and independent of it. They appear together in the instance settings dialog (Chapter 17):

- **Dead Time.** A short interval after a report during which further reports are suppressed, so a busy corridor does not saturate the bus. Typically a fraction of a second.
- **Hold Time.** How long the instance itself continues to regard the space as occupied after the last movement, repeating its report rather than falling silent.
- **Report Time.** How often that repetition is sent while occupancy persists.

These are encoded in coarse quantized units rather than as free-form durations — dead time in fractions of a second, hold and report times in steps of ten seconds and one second respectively — so a requested value snaps to the nearest step the standard can represent. Light-sensor instances add a hysteresis setting, expressed as a percentage with a minimum absolute value, which stops a sensor oscillating around its target as the light it is measuring is itself changed by the fixtures it controls.

Where a sensor's own hold time is long, it will keep a zone occupied regardless of what the zone's timeout says, since the zone never sees the silence. Setting a zone timeout shorter than the sensor's hold time achieves nothing.

C.8 Identity: GTIN and Product ID

A device carries two identifiers, and they do different jobs. Both appear as long strings of digits, which makes them easy to confuse, but the distinction between them underpins the whole of Chapter 18.

- **The GTIN** — a Global Trade Item Number — identifies the *model*. It is stamped into the device's firmware at manufacture and cannot be changed, and five identical drivers from the same batch all carry the same GTIN. DALI-2 requires it to be readable from the device. It answers the question *what kind of thing is this?* — which is why it keys the device catalog (Chapter 27), and why the system can know a device's capabilities before anyone has configured it.
- **The Product ID** identifies the *individual unit*. Unlike the GTIN it is assigned to the device rather than fixed at manufacture, and it is not of fixed length. It answers the question *which particular device is this?* — and it is what distinguishes one of those five identical drivers from the other four. Crucially it is held in two places: it persists in the device, and it is also a property of the corresponding fixture or input device record in the MCU database.

> **WHY IDENTITY MATTERS MORE THAN ADDRESS**
>
> A short address is positional and reassignable. Re-commission a bus and the addresses may land differently; add a device and the numbering shifts; replace a driver and the new one arrives with no address at all. Nothing about a short address is durable.
>
> Neither identifier is an address, and that is the point. Because the Product ID sits on *both* sides of the comparison — in the device, and against the MCU's record of that device — it is the one value that can be matched across the gap, while the short address is the thing free to vary. Matching on **Product ID** is therefore what lets the MCU recognize a device that has turned up at a different address from the one it expected, and the reconciliation mechanisms in Chapter 18 establish identity first and correct the addresses afterwards rather than trusting the addresses to begin with. Checking the **GTIN** does something different: it confirms the model is one the system knows, which is why that check runs before a newly discovered device can be dragged into a zone. The system establishes what a thing is, and which one it is, before it will let anyone say where it belongs.

C.9 How addresses get assigned

Control gear leaves the factory with no short address, which raises an obvious problem: how does a commissioning tool talk to a device individually in order to give it the address it needs to be talked to individually?

The answer is randomization. Every unaddressed device generates a temporary 24-bit random search address. The commissioning tool then performs a binary search over that 24-bit range — asking, in effect, "is anyone's search address below this value?" — halving the range each time until exactly one device remains. That device is given a short address and withdraws from the search. The process repeats until nothing answers. Sixty-four devices are resolved in a few hundred exchanges rather than sixteen million.

What randomization cannot do is tell anyone which luminaire in the ceiling received which number. Two mechanisms close that gap: the IDENTIFY command, which makes a chosen device flash so a person on the floor can see it, and physical selection, in which a device answers only while being physically manipulated. Chapter 18 describes how the MCU wraps both of these into its Identify and Assign workflow.

> **RE-ADDRESSING IS NOT A REPAIR**
>
> A full re-address discards the existing address assignment and generates a new one. Any mapping the MCU holds between its fixtures and the devices on the bus is invalidated at a stroke, and the network must be reconciled afterwards. It is a commissioning operation, not a troubleshooting one — reach for the reconciliation mechanisms in Chapter 18 first.

C.10 How MCU concepts map onto DALI

Much of the MCU's vocabulary is DALI's vocabulary with friendlier names. This table is the translation.

MCU concept	DALI concept beneath it
Network	One DALI bus
Fixture	One item of control gear, at a short address 0–63
Input device	One control device, at a control-device short address 0–63
Zone	Group 0–15 — the zone number is the group number
Scene within a zone	Scene 0–15 stored in each member device
Assigned role: DALI	DT6 LED module control gear (part 207)
Assigned role: 0-10V	DT5 DC-voltage conversion gear (part 206)
Assigned role: Relay	DT7 switching gear (part 208), driven outside the group
Product ID	A per-unit identifier, assigned to the device rather than fixed at manufacture
Device catalogue entry	GTIN — the model identifier held in the device firmware
Occupancy input	Instance type 303
Photosensor / light level	Instance type 304
Switch or button input	Instance type 301
Zone level	Arc power level 0–254 on the logarithmic curve

What has no DALI equivalent is just as instructive. DALI has no concept of a schedule, a calendar, a demand-response event, a user, a permission, a building, a tenant, or a day of stored history. It cannot coordinate two buses, let alone two sites. Everything in this manual above the level of a single network is the MCU supplying what the protocol deliberately leaves out.

> *Occupancy proposes; the schedule constrains; demand response trims. Each layer acts on the result of the one before it — and the IMS runs the whole thing locally whether or not anything above it is reachable.*

C.11 The IEC 62386 parts

Specifications, tender documents and manufacturer datasheets refer to the standard by part number. These are the ones likely to appear in a NICOR context; the standard has others.

Part	Subject
101	General requirements — system components. The bus, the electrical characteristics, the frame format.
102	General requirements — control gear.
103	General requirements — control devices. New in DALI-2; the reason sensors became interoperable.
104	Wireless and alternative wired media.
201	Fluorescent lamps (DT0).
202	Self-contained emergency lighting (DT1).
203	Discharge lamps (DT2).
204	Low-voltage halogen lamps (DT3).
205	Incandescent lamps, phase-cut control (DT4).
206	Conversion into DC voltage — 0-10V and 1-10V gear (DT5).
207	LED modules (DT6).
208	Switching function — relays (DT7).
209	Color control (DT8).
301	Push buttons.
302	Absolute input devices.
303	Occupancy sensors.
304	Light sensors.

A device may legitimately claim DALI-2 certification for the parts it implements and no others. When specifying replacement hardware, the part numbers are more informative than the phrase "DALI compatible", which has no defined meaning — see Chapter 27 on adding devices to the catalogue.

C.12 Glossary

Terms used throughout this manual. Where a term is NICOR's rather than DALI's, the definition says so and points to the chapter that covers it properly.

Term	Meaning
Application controller	The part of a DALI-2 system that makes decisions — reads events from control devices and issues commands to control gear. In a NICOR installation this role is performed by the IMS controller, not by any device on the bus.
Arc power	The DALI term for a light output setting, expressed on the 0–254 scale. Level 0 is off; 254 is maximum. The scale is logarithmic, so the raw value is not a percentage. See §C.5.
Area	NICOR term. A named part of a building holding one or more controllers, and the unit at which configuration is drawn and saved. See Chapter 8.

Term	Meaning
Assigned role	NICOR term. Whether a fixture is driven as DALI, 0-10V or Relay. Corresponds to the class of control gear behind it. See §C.10.
Backward frame	The 11-bit reply a device sends in answer to a query. Only one device may answer, which is why queries are addressed individually.
Broadcast	An address that every device on the bus responds to. The fastest way to command a whole network, and the only one that needs no commissioning.
Bus	The pair of wires carrying DALI signaling and bus power. One bus is one DALI network. See §C.2.
Bus power supply	The unit that energizes the bus, typically limited to 215 mA. Separate from the mains supply to the luminaires.
Control device	A device that senses or requests rather than emits — an occupancy sensor, light sensor or push-button unit. Standardized only from DALI-2 onwards. Appears in the MCU as an input device.
Control gear	A device that produces or switches light — an LED driver, a relay module, a 0-10V converter. Appears in the MCU as a fixture.
Controller	NICOR term. An IMS unit hosting one or more DALI networks and running the zone logic locally. See Chapter 15.
DALI	Digital Addressable Lighting Interface. The lighting control protocol defined by IEC 62386. See §C.1.
DALI-2	The second edition of IEC 62386, which standardized control devices and introduced mandatory identity data and certification. See §C.4.
Dead Time	A sensor timer that suppresses repeated reports for a short period after one has been sent, to limit bus traffic. Shown in the instance settings dialog. See §C.7.
Device type (DT)	A number identifying the class of control gear and the extended command set it supports — DT6 for LED modules, DT7 for switching, DT8 for color. See §C.11.
Extended fade time	A DALI-2 addition allowing fades shorter than the 0.7 s minimum of the original fade time scale.
Fade rate	The speed, in steps per second, at which a device moves when told to fade up or down without a stated destination.
Fade time	The time a device takes to move from its current level to a commanded one, selected from sixteen fixed values. See §C.5.
Fixture	NICOR term. One item of control gear as it appears in the MCU. Owned by exactly one zone. See Chapter 17.
Forward frame	The 19-bit command frame sent to a device or group. Takes roughly 16 ms on the wire, which is why individual addressing is slow. See §C.2.
Group	A DALI addressing construct, numbered 0–15, that lets one command reach many devices simultaneously. A NICOR zone is a DALI group. See §C.3.
GTIN	Global Trade Item Number. Identifies a product model, not an individual unit. Held in memory bank 0 of every DALI-2 device and used to key the MCU device catalogue. See §C.8.
Hold Time	A sensor timer governing how long an occupancy instance continues to treat a space as occupied after the last detected movement. A zone timeout shorter than this achieves nothing. See §C.7.
Hysteresis	A light-sensor instance setting, a percentage with a minimum absolute value, that stops the sensor oscillating around its target as the fixtures it controls change the light it is measuring.
Identify	A command that makes a device draw attention to itself, usually by flashing, so an installer can tie a short address to a physical luminaire. See §C.9.
IEC 62386	The multi-part international standard defining DALI. See §C.11.

Term	Meaning
IMS	NICOR term. The controller platform on which the MCU is built, and the component that runs zone logic autonomously at the site. See Chapter 2.
Input device	NICOR term. One control device as it appears in the MCU. Shared between zones rather than owned by one. See Chapter 17.
Instance	One sensing or input element within a control device. A single multi-sensor commonly presents an occupancy instance and a light-sensor instance. See §C.7.
Instance type	The number identifying what an instance does — 301 push button, 302 absolute input, 303 occupancy, 304 light sensor.
MASK	The reserved value 255, meaning "no change" or "not participating". A scene stored as MASK leaves that device where it is. See §C.6.
Master Control Unit (MCU)	NICOR term. The lighting management suite documented in this manual.
Memory bank	Addressable storage inside a DALI device. Bank 0 is mandatory in DALI-2 and carries the identity data. See §C.8.
MIN LEVEL / MAX LEVEL	Per-device limits on commanded output. A device told to go below its MIN LEVEL settles at MIN LEVEL rather than switching off. See §C.5.
Network	NICOR term. One DALI bus belonging to a controller, holding up to 16 zones, 64 fixtures and 64 input devices. See Chapter 15.
No Change	How the MCU renders MASK on a fixture's scene slider. Not an unconfigured scene — an instruction to stay at the current level when that scene is recalled. See §C.6.
Physical selection	A commissioning mode in which a device responds only when physically manipulated, used to isolate one unit without knowing its address.
POWER ON LEVEL	The level a device adopts when its mains supply is restored, without waiting to be told. See §C.5.
Product ID	NICOR term. An identifier assigned to an individual device, not of fixed length, uniquely distinguishing it from others of the same model. Persists in the device and is also held against the fixture or input device record in the MCU database, which is what allows the two to be matched. Distinct from the GTIN, which identifies only the model. See §C.8 and Chapter 18.
Randomization	The process by which unaddressed devices generate temporary 24-bit search addresses so a commissioning tool can isolate them one at a time. See §C.9.
Report Time	A sensor timer governing how often an occupancy instance repeats its report while occupancy persists. See §C.7.
Scene	A level stored inside each device against a number 0–15, recalled by a single command so that every device moves together. See §C.6.
SELV	Separated extra-low voltage. The classification of the DALI bus, which is why it may share a multicore cable with the mains supply.
Short address	A device's address on its bus, 0–63. Assigned at commissioning and reassignable, which is why the MCU matches on identity instead. See §C.3.
SYSTEM FAILURE LEVEL	The level a device adopts when the bus itself fails, so that a broken cable does not leave a space dark. See §C.5.
Zone	NICOR term. A group of fixtures driven as one, with its own four-state behaviors. Zone number is the DALI group number. See Chapter 4.
Zone type	NICOR term. A reusable template of zone settings sitting between the building and the zone in the inheritance chain. See Chapter 16.

Appendix D

OpenADR 3.1 Configuration

The MCU can act as a demand-response client to an external utility or aggregator, receiving grid signals over OpenADR 3.1 and dimming a building in response. This appendix is the field-by-field reference for the connection settings. The behaviour those settings drive — what the signals mean and how a building acts on them — is Chapter 21 §21.8; this is the sheet to have open while filling the form in.

D.1 The parts, in one model

OpenADR describes a conversation between two roles. The **VTN** — Virtual Top Node — is the utility or aggregator's server, which decides when demand response is needed and publishes events. The **VEN** — Virtual End Node — is the client that listens and acts. When a building is configured here, the gateway's service for that building *is* the VEN.

The settings on this screen fall into four jobs: where the VTN is and how to reach it, how to authenticate to it, how this VEN identifies itself, and how it is notified of events. The screen groups them the same way, and so does this appendix.

> **LICENSING**
>
> The ADR configuration entry on the Building Properties menu is present only for clients whose licence includes ADR (Chapter 25). Without the licence the entry is disabled, and the building cannot act as a VEN.

D.2 Reaching the VTN

The VTN provider supplies every value in this group. Together they compose the base URL the VEN calls.

Field	Meaning
VTN Host	The hostname or IP address of the VTN server.
VTN Port	The port it listens on.
VTN Base Path	Any path segment the VTN is served under, appended to host and port. Some VTNs serve at the root and need this left blank; others sit behind a versioned path. Follow the provider's documentation exactly — a wrong or missing base path is a common cause of every call returning *not found*.
Use TLS	Whether the VEN connects over HTTPS rather than HTTP. **Enable it for any real deployment** — the token and event traffic are sensitive. Plain HTTP is for local testing only.

D.3 Authenticating to the VTN

OpenADR 3.1 uses OAuth2 client-credentials. The VEN exchanges a client ID and secret for a short-lived bearer token at the VTN's token endpoint, and presents that token on every request.

Field	Meaning
Token Url	The OAuth2 token endpoint, relative to the VTN base URL — commonly `auth/token`. This is where the client credentials below are exchanged for a token.
Auth Scope	The OAuth scope to request, where the VTN requires one. Left blank when it does not. The provider states whether a scope is needed and what value to use.

D.4 Identifying this VEN

These four values tell the VTN who is connecting and which programme to follow. The credentials here are the ones the VTN issued for this building's VEN account.

Field	Meaning
Client Id	The OAuth2 client identifier for this building's **VEN account**. Not a business-logic or administrative account — see the warning below.
Client Secret	The secret paired with the client ID. Treat it as a password: it is all that stands between anyone and the ability to impersonate this building's VEN.
Program Id	The programme this VEN subscribes to, given as the **VTN-assigned UUID** — a value like `907e7205-8992-4213-a408-25b1bc1adb8e`, not the programme's friendly name. See the warning below.
VEN Name	The name this VEN reports to the VTN. It identifies the building on the VTN side and appears in the provider's logs, so make it recognisable.

PROGRAM ID IS THE UUID, NOT THE PROGRAMME NAME

This is the single easiest field to get wrong, and it fails silently. A programme has a human-readable name and a separate **id** that the VTN assigns when the programme is created. The VEN scopes its event reads by that *id*.

Enter the *name* here instead of the UUID and the VTN matches nothing: the VEN authenticates cleanly, polls without error, and simply never sees an event. There is no failure message to follow — the building just never responds to demand response. If a correctly-configured-looking VEN receives nothing, check this field first.

USE THE VEN ACCOUNT, NOT A BUSINESS-LOGIC ONE

A VTN typically issues separate credentials for reading events (the VEN) and for authoring programmes and events (a business-logic account). The Client Id and Secret here must be the VEN account. Business-logic credentials may authenticate but will not behave as a VEN is expected to, and using them is both wrong and needlessly over-privileged.

D.5 How the VEN is notified

There are two ways the VEN learns that an event has appeared or changed: it can **poll** the VTN on a timer, or it can receive **MQTT** push notifications. They are not exclusive — the common and most robust arrangement is to run both.

Polling

Field	Meaning
Enable polling	Whether the VEN periodically asks the VTN for the current event model.
Poll Interval	How often it asks, in seconds. This value bounds how quickly the building reacts, and it matters more than it looks: a demand-response event can step through several levels *within a single event*, and the VEN only re-evaluates on a poll or a notification. A poll interval longer than the shortest level step means transitions inside an event can be missed. Set it at or below the finest granularity the programme uses.

MQTT

Where the VTN offers MQTT push, notifications arrive as events are created, updated or deleted, without waiting for the next poll. This makes the building react promptly to a new event while polling continues to catch the level transitions inside it.

Field	Meaning
Enable MQTT	Whether the VEN subscribes to the VTN's MQTT broker for push notifications.
MQTT Host	The broker's hostname or IP address. Often the same machine as the VTN, but not necessarily.
MQTT Port	The broker port. 1883 is the plain-MQTT default; a TLS broker usually listens on 8883.
Use TLS	Whether the broker connection is encrypted. As with the VTN connection, enable it for any real deployment.
User	The username for the broker, where it requires authentication.
Password	The paired password. Security-sensitive, like the client secret.

A PUSH TELLS YOU SOMETHING CHANGED — IT DOES NOT CARRY THE SCHEDULE

An MQTT notification fires when an event object is created, updated or deleted. It does not fire at each level boundary inside an event. So a notification prompts the VEN to re-read the current model, but the None →Level1 →Level2 →Level3 steps within an event happen as wall-clock time crosses each boundary.

This is why polling is kept on even when MQTT is enabled, and why the poll interval still has to be short enough to catch the steps. MQTT makes the building notice a new event quickly; polling is what keeps it in step through the event's life.

D.6 Report confirmation

Field	Meaning
Confirm events	Whether the VEN sends acknowledgement reports back to the VTN, confirming it received an event and is acting on it. Off by default. Enable it only where the programme asks for confirmation — some VTNs require it, many do not, and it is an opt-in because an unwanted report is just noise on the VTN.

D.7 What the building does with a level

OpenADR's SIMPLE signal carries a value from 0 to 3, and the VEN maps it straight onto the building's own demand-response levels — the same levels configured per zone type in Chapter 21 §21.3.

Field	Meaning
SIMPLE 0	Normal. No demand response — the building runs unconstrained.

Field	Meaning
SIMPLE 1	The building's demand-response Level 1 reduction.
SIMPLE 2	Level 2.
SIMPLE 3	Level 3, the most severe reduction.

The building acts on these automatically only while its demand response is in **automatic mode**. A manual override set on the building (Chapter 21 §21.4) takes precedence over the incoming signal — the ADR client keeps receiving events, but the override is what the zones follow until it is cleared. When no event is active, or when the signal is 0, the building sits at no reduction.

> The ADR client is one more source feeding the building's existing demand-response machinery, not a parallel one. The signal chooses a level; everything downstream of that — which zones dim, by how much, and what a manual override does to it — is exactly as Chapter 21 describes. *The grid asks; the building's own rules decide what that means room by room.*

PART V · APPENDICES

Appendix E

Troubleshooting and Frequently Asked Questions

A lookup rather than a tutorial. Find the symptom, follow the checks in the order given, and go to the chapter named for the detail. The second half answers the questions that come up most often about why the system behaves as it does — several of which are not faults at all, but designed behavior that surprises people the first time they meet it.

E.1 Before anything else

Chapter 32 §32.6 sets out a diagnostic order, and it is worth internalizing because it saves more time than any individual answer in this appendix: check licensing and permissions, then the gateway connection, then the controller's connection and mode, then whether the change reached the hardware, then the device model, and only then the zone itself. Working downward rules out whole layers at a time.

> **THE LIGHTS ARE PROBABLY FINE**
>
> The IMS runs zone logic locally and autonomously. Occupancy is evaluated, timeouts count down and fixtures respond whether or not the gateway is reachable, whether or not the DataManager is running, and whether or not anyone is logged in.
>
> So a report of "the system is down" usually describes a loss of visibility rather than a loss of lighting. Red zone markers across a plan, "Live status unavailable" banners, a building that cannot be opened — in every one of those cases the rooms themselves are almost certainly lit correctly. Establishing that early changes the urgency of everything that follows.

E.2 Nothing appears to be working

Symptom	What to check, in order
Nobody can log in	The DataManager. Logins depend on it; running gateways carry on without it, but nobody can sign in. See Chapter 28.
"Live status unavailable" across a whole building	The gateway connection. Configuration remains editable; live status does not. The lighting is unaffected. Chapter 29.
A feature is missing for everyone	Licensing (Chapter 25). If it is missing for one person only, permissions (Chapter 23).
Zone markers red across the plan	The parent controller is not connected, so the MCU cannot report state. Appendix B §B.2 — this is a visibility problem, not a fault.

E.3 A change will not take effect

This is the single most commonly reported problem, and it has one overwhelmingly likely cause.

CHECK THE CONTROLLER'S NETWORK MODE FIRST

A controller left in **Read only** after commissioning will never receive configuration. It stays connected, reports status normally, and responds to direct and quick-test commands — so everything looks healthy while nothing you save ever arrives.

It presents exactly as "the configuration mysteriously does not apply", and it takes seconds to rule out. Chapter 15 §15.5.

Symptom	What to check, in order
Saved, but nothing changed	Controller network mode (Chapter 15 §15.5); then building update mode (Chapter 13); then whether the area was actually saved back to the DataManager, which is what advances the revision the gateway notices (Chapter 19 §19.2).
Some controllers updated, others did not	The mode of the controllers that did not. The two-tier policy is evaluated per controller.
No synchronization sequence in the gateway report after saving	The change was not delivered. Look at the gateway connection and the controller mode rather than at the configuration. Chapter 32 §32.3.
A setting reverts or is ignored	Inheritance. A value marked inherited comes from the parent regardless of what is typed at a lower level. Chapter 5 §5.2.

E.4 The wrong lights respond

Symptom	What to check, in order
The wrong fitting responds to a zone	Addressing. The MCU's model and the bus disagree about which device sits where. Chapter 18.
Nothing at all responds in one zone	Addressing first, then whether the controller is connected, then whether the fixtures are in the zone at all.
Part of a zone will not dim	Look for switch symbols on the plan. Relay-role fixtures sit outside the DALI group and can only be on or off. Appendix B §B.5 and Chapter 16.
One luminaire behaves oddly after replacement	Scene levels live in the device. A replacement arrives knowing nothing about the scenes it is expected to join. Chapter 17 §17.4.
A fixture will not go below a certain level	Its Minimum. A driver commanded below its minimum settles there rather than switching off. Chapter 17 §17.3.

E.5 Occupancy and timing

A ZONE THAT NEVER TIMES OUT IS USUALLY THE SENSOR, NOT THE ZONE

An occupancy instance has its own **Hold Time**, during which it continues reporting the space as occupied. If that is longer than the zone's timeout, the zone never sees the silence it is waiting for and the timeout achieves nothing.

Setting a shorter zone timeout will not help. Chapter 17 §17.7, and Appendix C §C.7 for why occupancy sensors work this way at all.

Symptom	What to check, in order
A zone is frozen in one state and ignores everything	Look for a flashing padlock over the zone on the plan — that is a BACnet client holding it. Occupancy, inputs and timeouts are all inactive while a lock is held. Appendix A §A.6.
A zone never times out	A BACnet lock first, then the sensor's Hold Time as above, then the state machine's destination states — a state whose timeout leads back to itself will never progress. Chapter 16.
A zone times out with people in it	Sensor coverage, then the Preliminary and Occupied timeout values. The zone can only know what it is told.
A zone will not become occupied	Whether an occupancy input is associated with it, and whether the instance is enabled. Chapter 17 §§17.6–17.7.
A button does nothing	The action matrix for that event in the zone's current state — "No Action" in one column is deliberate and easy to overlook. Chapter 16. Trigger the event from the canvas to test without walking. Chapter 17 §17.10.
Lights come on at full after a power cut	The device's power-on level, held in the driver. Nothing above it is involved. Appendix C §C.5.

THREE THINGS CAN HOLD A ZONE AWAY FROM ITS OWN LOGIC

A **BACnet lock**. A write from a building management system takes the zone over entirely, and only a write of 5 — "Release to IMS" — gives it back. A client that disconnected without releasing leaves the zone locked indefinitely. Appendix A §A.6.

A **demand response override**. Set at the building and visible in its Override set row; Auto clears it. Chapter 21 §21.4.

The zone being **disabled**. Its enabled flag is off, so it does nothing at all. Chapter 16.

None of the three reports a fault, because none of them is one — each is something working exactly as designed. That is what makes them hard to find: every part of the lighting system is behaving correctly and the room is still wrong.

Each does announce itself somewhere, though, and all three checks are quicker than opening the zone. A lock shows a flashing padlock over the zone on the plan; a demand response override shows in the building's Override set row; a disabled zone shows its enabled flag cleared. Look at those three before investigating the zone itself.

E.6 Daylight harvesting and demand response

Symptom	What to check, in order
Lights visibly breathe or hunt	Widen the target tolerance before lengthening the adjustment interval — tolerance addresses the cause, the interval only makes the same corrections less often and larger. Chapter 21 §21.6.
A building is unexpectedly dim	The demand response status rows. A level in force with no external request means an override was left set; the fix is Auto, not the zone configuration. Chapter 21 §21.4.
One zone is dimmer than its target	Sensor placement. A sensor with a strong view of the fixtures it controls measures its own output rather than the daylight. Chapter 21 §21.5.
Demand response does nothing	Licensing first (Chapter 25); then whether the zone type is set to No action at that level. Chapter 21 §21.3.

E.7 Devices and addressing

Symptom	What to check, in order
Count mismatch reported on a network	Do not run product-ID matching or an import. Scan to see what is actually present, then reconcile. Chapter 18 §§18.3 and 18.11.
A scan stops with a list of numbers	Unknown GTINs. Those models must be added to the device catalogue before anything can be imported. Chapter 18 §18.8, Chapter 27.
Fixtures and Input Devices tabs are locked during discovery	Zones must be imported first, and the software enforces it. Chapter 18 §18.8.
An import stops with an error about missing objects	The controller holds objects the MCU does not, and an import can never create them. Scan first. Chapter 18 §18.7.
One device will not pair during a product-ID match	Its Product ID — blank or wrong on the MCU record. Chapter 17 §17.2, Chapter 18 §18.10.
Everything on a network is wrong after re-addressing	Re-Address ALL discards the whole assignment. Reconcile rather than re-addressing again. Chapter 18 §18.9.

E.8 Energy, history and backups

Symptom	What to check, in order
Energy figures look implausible	The driver load recorded against the fixtures. Energy is estimated from load-type power curves, never metered, so a wrong load produces plausible but incorrect numbers. Chapter 17 §17.2, Chapter 10.
Historical energy has disappeared	Data older than the DB-history period is archived to long-term storage rather than deleted. Chapter 31.
Zone state history only goes back a day	By design — it is held on the gateway on a rolling 24 hours and never enters the database. Chapter 31.
A restored backup is missing things	A database backup covers the database only — not long-term storage and not the audit log. Chapters 28 and 31.

E.9 Questions that come up repeatedly

Why does a level of 50% not look like half the light?

It does, if it is the percentage the interface shows — that figure is actual light output. What is not half is the raw DALI value beside it: the scale is logarithmic, so fifty per cent of output sits at around level 229 rather than the midpoint of 127. Read the bracketed number as a position on a curve, never as a percentage. Appendix C §C.5.

Why is there a timeout at all? Why not just detect that the room is empty?

Because no sensor can report emptiness. An occupancy instance reports one thing — movement detected — and there is no message for "nobody is here", since absence is not an event. Emptiness has to be inferred from the passage of time since the last report, and that inference is exactly what a timeout is. Appendix C §C.7.

Why only sixteen zones and sixty-four fixtures?

Those are DALI's limits, not NICOR's. A bus has sixteen group addresses and sixty-four short addresses for control gear, with a separate sixty-four for control devices — so a network holding sixty-four fixtures can still hold sixty-four input devices. Appendix C §C.3.

Why does a new controller arrive in Read-only?

Because at the moment it is added, the MCU's model of it is empty or half-built while the controller may be carrying a fully commissioned installation. The MCU's model is the poorer of the two, and pushing it would overwrite working configuration. Read-only removes the possibility until the two have been reconciled. Chapter 15 §15.5.

Does the lighting stop if the server goes down?

No. The IMS runs the site locally. Losing the gateway costs live status; losing the DataManager costs logins and history. Neither stops a room responding to occupancy. Chapter 2.

Why does moving a fixture to another zone need the controller connected?

Because a zone is a DALI group and membership is held on the driver itself. Moving a fixture rewrites group membership on hardware in the ceiling, so the change has to reach the bus to mean anything. Chapter 17 §17.9.

Why did importing from a controller pin hundreds of overrides?

The two DALI-settings checkboxes on the import confirmation were ticked. Left clear — their default — device-specific values are ignored and the properties inherit normally; ticked, they arrive as explicit per-device overrides that then ignore anything set above them. Chapter 18 §18.7.

A zone marker is red. How bad is it?

Usually not bad at all. On a zone marker red means the MCU cannot determine the state, almost always because the controller is unreachable. The zone itself is very likely detecting occupancy and driving its fixtures exactly as configured. Appendix B §B.2.

> *Most reported faults are one of three things: a controller left in Read-only, a sensor hold time longer than the zone timeout it is supposed to obey, or a loss of visibility mistaken for a loss of lighting. Rule those out first.*