



Integration between the Ksenia lares 4.0 alarm panel and the LogicMachine.

Versione italiana: [manuale.md](#).

The app connects a **Ksenia lares 4.0 alarm panel** to a **LogicMachine**. The LogicMachine reads the state of the system continuously – partitions, zones, inputs, outputs, temperatures – and can control it: run scenarios (arming included), bypass a zone, turn an output on or off.

You need **one licence per LogicMachine**.

1. What you need before you start

IP address of the panel	the address of the lares 4.0 on the network, not the LogicMachine's
A user code (PIN) of the panel	the app logs in with it
The package file (.ipk)	we provide it
MAC address of the panel (optional)	12 characters, from the label or the Ksenia software. The panel does not use it for the login: it can be left empty

The LogicMachine must **reach the panel over the network**: the app connects with `wss`, on port **443** of the panel.

⚠ Choose the PIN carefully. The app controls the panel as *that user*, so on the panel every action is recorded as done by that user, and the app can only control that user's partitions. Create a user dedicated to the integration rather than using the owner's.

Tested on a lares 4.0 16IP with firmware 2.6.22.

2. Installing the app

From the LogicMachine interface:

1. **App Store → Install from file**
2. choose the `.ipk` file
3. leave the *Signature file* field **empty** and tick **«Allow to install package without signature»**
4. confirm

The warning about the missing signature is normal and does not mean there is a problem: it only means the package is not signed by the LogicMachine manufacturer.

After the installation the app appears in the list of applications as **Logic2Ksenia**, with its version number next to it (for example `1.0.0`). Open it from there, or at:

`http://LOGICMACHINE-ADDRESS/apps/data/logic2ksenia/`

There is nothing else to install: no script to paste, no library to add. The app starts the service that holds the connection by itself, and restarts it every time the LogicMachine restarts.

The language is chosen at the top right, **Italiano** or **English**. It applies to **everyone** who opens the app on that LogicMachine, not only to whoever changes it: it is decided once, at installation time.

3. The licence

The licence is bought on **agatastore.it**, the Agata Automation shop. It is **perpetual** and valid for **one LogicMachine**. The first time it is opened, the app shows a box with the **Fingerprint of this device** – a string such as `1m:6110DC18180B21D7`.

1. buy the licence on agatastore.it
2. an e-mail arrives with the link to request it
3. in the app press **Copy** next to the fingerprint, and paste it into the page of the link
4. once the payment is confirmed the licence arrives by e-mail, usually within a few minutes: it is a text file
5. open it with a text editor and copy **all** of its content
6. paste it into the box in the app and press **Save the licence**

When the licence is in place **the box disappears**: that is the sign that everything is in order. If it stays visible, its text says what is missing.

What it says	What it means
missing	it has not been installed yet
unreadable	the pasted text contains no fingerprint and signature: copy the file again, in full
for another device	it is the licence of another LogicMachine: you need the one for yours
not valid	the file was copied badly, or modified: copy it again from the e-mail, or write to us
not verifiable	the app works, but it cannot check the licence. Send us the fingerprint and report it to us

With *missing*, *unreadable*, *for another device* or *not valid* **the app does not connect to the panel**.

To replace a licence with another one, just paste the new one. To go back to the «*missing*» state – it is only useful to check that the connection is really refused without a licence – there is **Discard the licence**, among the buttons next to *Diagnostics*.

 **The licence is bound to that LogicMachine.** If the LogicMachine is **replaced**, the old licence is no longer valid and a new one is needed. If the licence is lost instead, write to us: we will send it again.

4. Connecting the panel

In the fields at the top:

- **IP address** of the panel

- **MAC address** of the panel, optional
- **User PIN**
- **Access:** *User*, which is the normal connection (for *Supervisor* see below)

Then **Connect**. Within a few seconds the indicator next to the button reaches **Connected**, and the cards fill in with the state of the system.

The connection belongs to the LogicMachine, not to the page. When you close the app or the browser the connection stays up, and the LogicMachine objects keep updating. It only stops with **Disconnect**; the indicator then says **Bridge stopped**.

The PIN is typed only once. It is stored on the LogicMachine, and from then on the connection restarts by itself, even after a reboot: it does not have to be typed again. The PIN is **never sent back to the browser** and does not appear in any log.

If the login is refused, the data is **not** saved: correct it and try again.

The PIN is encrypted, and the key is the LogicMachine itself. It is not stored in clear text: it is sealed with a key derived from the CPU identifier of the machine, which a project backup does not contain. There is one practical consequence, and it is worth knowing in advance: **if the project is restored on a different LogicMachine, the app asks for the PIN again.** This is not a fault — it is what stops a backup from carrying it away.

Supervisor access

The **Access** drop-down has two entries. **User** is the normal connection. **Supervisor** uses the reserved connection that the panel keeps for a supervisor: it always finds a free slot, even when the others are taken.

Before choosing it, on the panel:

1. under **IP devices** → **Supervisors** (*Dispositivi IP* → *Supervisor*), configure a supervisor with the **MAC address of this LogicMachine**. The app shows it under the drop-down when you choose *Supervisor* («*MAC on the Ksenia: ...*»): copy it from there. It is **not** the panel's MAC, the one in the field at the top;
2. leave *Ignore* off.

Then, in the app:

1. **connect at least once as User**, with the PIN of a panel user. This checks the PIN and saves it;
2. choose **Supervisor** and press **Connect**. The PIN field is disabled: the LogicMachine derives the supervisor PIN by itself from its own MAC address.

Commands are sent with the saved user PIN. The panel checks that user's partitions and records the events under that user's name.

 **If you change that user's PIN on the panel, connect again as User with the new PIN.** The supervisor connection does not use that PIN, so it keeps succeeding, but commands are refused. A refused command is not repeated by itself.

If the supervisor connection is refused, the first thing to check is that the MAC configured on the Ksenia is the one shown by the app. *Diagnostics* reports it on the line `mac della LM (supervisore)`.

5. Bringing the state onto LogicMachine objects

This is what makes the integration useful: every property with the **pencil** icon can be linked to a LogicMachine object, which is then updated in real time.

Press the pencil, choose or create the group address, save. For some properties the window also lets you choose the **type** of the object (for example *Boolean* or *1 byte*): the note under the choice says what each value means.

What can be linked

Element	Properties
Zone	<i>Idle / active · Alarm · Bypass · Command: bypass the zone</i>
Partition	<i>Arming(boolean, or 1 byte: 0 disarmed, 1 total, 2 partial) · Alarm</i>
Output	<i>State · Command: switch the output</i>
Scenario	<i>Command: run the scenario · Category(fixed value) · Zones blocking arming · Arming result · Command: arm anyway</i>
System	<i>System arming · System alarm · Tamper · Fault · Alarm memory · Tamper memory · Fault memory · Notice · Indoor temperature · Outdoor temperature</i>
Sensors	<i>Temperature · Humidity · Light(DOMUS sensors)</i>

The seven system signals (from *System alarm* to *Notice*) can be created as a **boolean** — 1 when there is something — or as **text**, with the list of causes.

Virtual or Standard

In the pencil window you choose where the group address is created:

	Addresses	What it means
Virtual (<i>the default</i>)	from 32/0/0 to 63/7/255	the object lives only inside the LogicMachine. It does not touch the KNX bus
Standard	from 0/0/1 to 31/7/255	the object goes onto the KNX bus of the installation : the other KNX devices can see it, and write to it

Unless there is a specific reason to bring the panel state onto the bus, stay on **Virtual**. Choose Standard only knowing what is on that bus, and that the address is not already used by another device.

Command objects

Some properties work in the other direction: writing to the LogicMachine object **controls the panel**. There are three actions:

You write to	And the panel	How the object behaves
the scenario (<i>Command: run the scenario</i>)	runs the scenario — this is also how you arm and disarm	it is a pulse : you write 1, and the object goes back to 0 by itself
the zone (<i>Command: bypass the zone</i>)	bypasses the zone (2), bypasses it until the next disarm (4), or includes it again (0)	it stays on the written value, and follows the panel when the state changes by itself
the output (<i>Command: switch the output</i>)	turns the output on (1) or off (0)	it stays on the written value, and follows the panel when the state changes by itself

The difference between a pulse and a value that stays matters: an object that *stays* is a **standing request**, and until it is set back the zone remains bypassed.

When the state changes without a command from you, the command object follows it. A zone bypassed with 4 comes back by itself when the system is disarmed, and its object returns to 0; a zone bypassed or an output switched off from the keypad moves the object to the new value. So the visualization button always starts from the real state: writing 4 again bypasses the zone again. This update **sends no command** to the panel, and if you write something at the same moment, what you wrote wins.

To show the state, use the state object, not the command object. In the visualization the button writes to the command object (*Command: bypass the zone*), but the indicator must read the state object (*Bypass*): it is the only one that says what the panel really did. The app does not send a command that would change nothing — including a zone that is already included — and if after a command the panel announces a state different from the one requested, it writes a warning in the LogicMachine log, without repeating the command.

A command reaches the panel within a few seconds: the app checks the command objects about every 5 seconds.

⚠ The LogicMachine *Set value* editor respects the *minimum value* of the object's visualization parameters: if it is 1, writing 0 — including the zone again — is not possible. Leave it at 0.

🔴 **Know this before creating them, and tell the owner of the system.** Whoever can write to that object — a visualization, a scene, another system connected to the LogicMachine — **can arm or disarm the system without knowing the PIN**. The app asks for confirmation every time one is created. It is a feature, not a defect: use it knowingly.

⚠ **And bypassing a zone is the only one of the three that makes no noise.** Arming sounds, an output can be seen; a bypassed zone cannot — no siren, no notification, and the system **stays armed** with a gap that announces nothing. Whoever creates this object must know it, and so must the owner.

Arming with open zones

If a zone is open, the panel **refuses** the arming scenario and says which zones prevent it. Three objects of the scenario let you decide, case by case, to arm anyway:

Object	Who writes it	What it contains
<i>Zones blocking arming</i>	the app	the IDs of the zones of the last refusal (1,3), or empty. It empties by itself after 60 seconds
<i>Arming result</i>	the app	ok , or the reason why it did not arm (refused:1,3 , bypass-failed:3 ...)
<i>Command: arm anyway</i>	you	1 bypasses until the next disarm only the zones in the list, checks that the panel has bypassed them, and runs the scenario again

Zones bypassed this way **come back by themselves** when the system is disarmed: no forgotten gap is left. If a step fails, the app stops without arming and does not retry by itself.

 **The *arm anyway* command arms the system with open zones.** The warning above applies: whoever can write to that object does so without knowing the PIN.

With logic2visu

If **logic2visu** is also on the LogicMachine, its Visu pages show the alarm by themselves: Logic2Ksenia tells it which objects you have linked, with the names from the panel and the confirmation questions. **There is nothing to configure:** what you have linked appears. How the pages are generated and updated is explained in the logic2visu manual.

In the generated pages every command asks for confirmation with a question that names the action and the element («*Bypass zone "Door" until disarm?*»). A refusal for open zones opens a window with the names of the zones and the *Arm anyway* button.

6. Who can see what

Anyone with access to the LogicMachine sees the state of the alarm without typing any PIN. That is the point of the integration — showing the system inside the supervision — but the customer must be told before, not after.

Access control is the LogicMachine's: users, passwords and, if needed, remote access through a VPN. **Do not expose the LogicMachine on the internet with port forwarding.**

7. If something goes wrong

The **Diagnostics** button produces a full report. When you write to us, **send it in full:** it already contains nearly every answer we would otherwise have to ask you for. The report itself is in Italian.

Symptom	First thing to check
it does not connect	the panel IP address; that the LogicMachine can reach the panel on port 443; that the licence is valid
the login is refused	the user PIN
the login as supervisor is refused	that the MAC under <i>IP devices</i> → <i>Supervisors</i> is exactly the one the app shows under the drop-down
it connects and disconnects	unstable network between LogicMachine and panel
the state is frozen	the log panel says for how long; send the Diagnostics report
an object does not update	open the link again and check the group address
a command is refused with WRONG_PIN	the saved PIN is no longer the panel's: connect again as <i>User</i> with the right PIN
a command does not go out, and the log says « <i>NON mandato</i> »	the log says why: for example the zone was already in the requested state
the app asks for the PIN again, after a project restore	this is intended: the PIN is encrypted with the LogicMachine it was saved on (§4). Type it again
« <i>for another device</i> »	it is the licence of another LogicMachine

The **Connection log** at the bottom of the page tells what is happening in real time. The *Verbose* box adds the technical details, and *Log to the LM* also copies the messages into the LogicMachine log. Some technical lines in the log stay in Italian.

8. Uninstalling

In this order, or data stays on the LogicMachine that nobody removes afterwards:

1. in the app: **Disconnect**
2. in the app: **Remove the data** — it deletes parameters, PIN, links and licence. If it answers that the data was only partly removed, send us the message: it names what is left
3. from the LogicMachine interface: remove the package

The **objects created on the LogicMachine are not deleted**: they are LM objects, they may have ended up in a visualization or a scene, and deleting them would destroy someone else's work. They stay, inert.

9. Updating the app

Install the new **.ipk** over the existing one, as in §2: **App Store** → **Install from file**. There is no need to remove the installed version first.

Parameters, PIN, links and licence are kept: after the update the app starts already configured, **and the licence stays valid**, because it is bound to the LogicMachine and not to the version of the app.

⚠ **Do not press *Remove the data*** to update: it would delete everything, licence included.

Versions

The number follows **Semantic Versioning**, MAJOR.MINOR.PATCH :

1.0.0 → 1.0.1	fixes. Update without thinking about it
1.0.0 → 1.1.0	new features, everything else keeps working as before
1.0.0 → 2.0.0	something changes in a non-compatible way: read the release notes before updating

The installed version is shown in the list of applications, and in full – with the commit and the build date – in the **Diagnostics** report. It is the first thing to give us when you report something.

What the licence promises, said plainly

The licence binds the app to **one** LogicMachine, and it is a **contractual** constraint. Whoever has administrator access to their own LogicMachine can technically bypass the check: no product in this category can prevent that. What the licence guarantees is that **nobody can produce valid licences** for other devices.