ToddySoft Connect Modbus Driver - Ignition 8.3

Ignition 8.3 module for connecting to **Modbus TCP** devices — PLCs, meters, drives, I/O blocks and gateways. Coils, discrete inputs, input registers and holding registers are addressed directly, with a type suffix that tells the driver how to decode the bytes.

The device configuration uses a custom configuration form: settings that do not apply to what you have selected stay out of your way, and the fields that carry a security consequence say so where you read them.

> **Modbus has no tag list, and this is a protocol fact rather than a gap in the driver.** Nothing on the wire enumerates addresses, so you write addresses by hand from the device's Modbus map. Every other ToddySoft Connect module browses a controller for its symbols; this one cannot, and no setting will change that. Devices do not announce themselves either, but they can be found: see Finding devices on the network.

Installation

> **This module is for Ignition 8.3 only.** It is built against the Ignition 8.3 SDK and will not run on any other major version. For Ignition 8.1, install the separate `…-8.1.modl` build instead — the two are not interchangeable, and each gateway needs the build that matches it.
>
> **Ignition 8.0 and earlier are not supported and cannot be.** The module is built on the device-driver API that Ignition introduced with 8.1, so on 8.0 the gateway fails to load it with `NoClassDefFoundError: … opcua.server.api.AbstractDeviceModuleHook`.
>
> Both builds carry the same name in the gateway's module list, so tell them apart by the file you installed or by the module description, which names the Ignition version.
>
> **Ignition's OPC UA module must be installed and running.** This is an OPC UA device driver: it registers with Ignition's own OPC UA server, and every tag it exposes is served through it. With that module absent the gateway cannot load this one, and the log fills with `ClassNotFoundException` / `NoClassDefFoundError` naming `com.inductiveautomation.opcua…` classes. The OPC UA module ships with Ignition and is installed by default, so in practice this bites only on a gateway where it has been removed or disabled. This module declares the dependency as **required** — a mechanism 8.3 has and 8.1 lacks — so Ignition does not attempt to load it at all, and reports the reason instead of failing with a stack trace. In **Diagnostics > Logs** you will see:
>
> `Module "com.toddysoft.connect.…" depends on "com.inductiveautomation.opcua", which is either not installed or inactive on this gateway.`
>
> This is the same handling Ignition's own drivers get in the same situation.

1. Open the Ignition Gateway web interface and navigate to **Platform > System > Modules**.
2. Click **Install or Upgrade Module**.
3. Select the `ToddySoft-Connect-Modbus-Driver-8.3.modl` file and upload it.
4. Accept the license / certificate prompt to continue the installation.
5. **Restart the gateway.** Ignition 8.3 does not load a newly installed or upgraded module while it runs: the modules list shows it as *pending restart*, and a **Gateway Restart Required** banner at the bottom of the Modules page offers **Restart**

Gateway. Several modules can be installed or upgraded first and taken over with one restart. Plan it: the restart briefly interrupts every device and client on the gateway.

6. After the restart, check that the module appears as **ACTIVE** in the modules list.

Before you start: what you need to know about the device

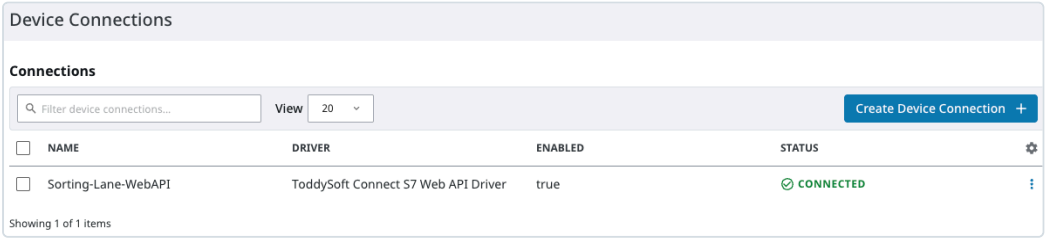
- **Its Modbus map.** Which registers hold what, and in which representation. There is no way to ask the device, so this comes from its documentation.
- **Its unit identifier** (slave id), if it is not `1`.
- **Its byte order** for anything wider than one register. Modbus defines the byte order inside a register (high byte first) but not the order of the registers in a wider value, so vendors differ. See *Default Byte Order*.
- **TCP port 502** reachable from the gateway host, unless the device listens elsewhere.

There are no credentials: Modbus is unauthenticated. Anything on the network that can reach the port can read and write it, which is worth knowing when deciding where the gateway sits.

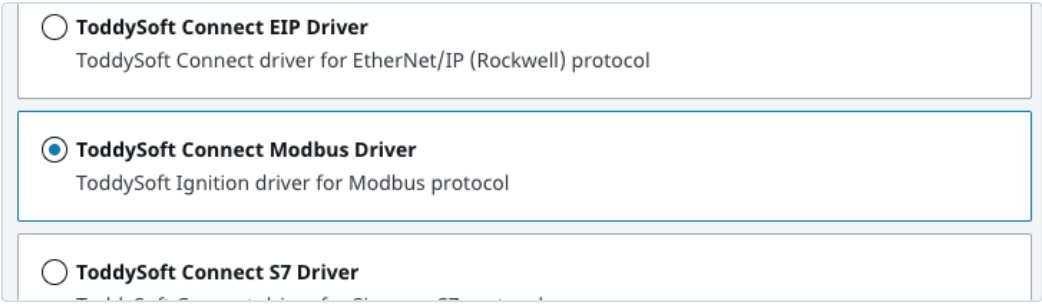
> **This module speaks Modbus TCP only.** On Ignition 8.3 both the module and the device type are named *ToddySoft Connect Modbus Driver*; on Ignition 8.1 the device type spells it out as *Modbus TCP Driver*. The connection is the same either way. The underlying driver also implements Modbus RTU and ASCII over serial, UDP and TLS, but the Ignition module emits a `modbus-tcp://` connection and exposes no transport selector, so those are not reachable from here. A serial device needs a Modbus gateway in front of it.

Creating a Device Connection

1. Go to **Connections > Devices > Connections**.

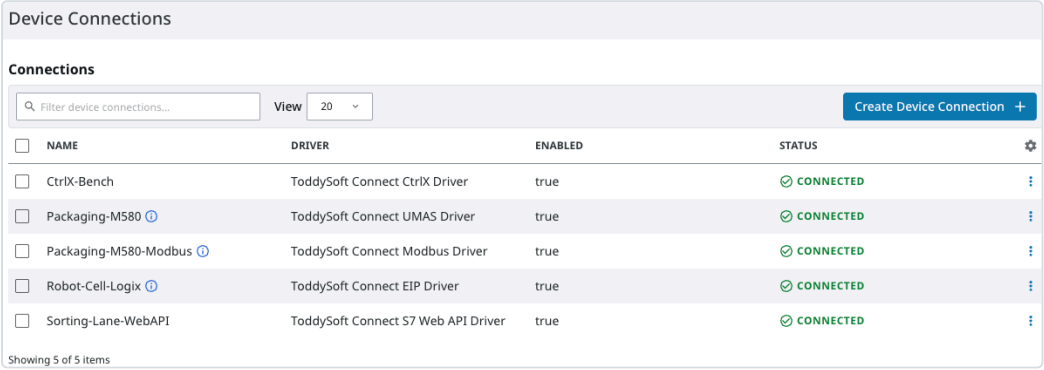


2. Click **Create Device Connection** and select **ToddySoft Connect Modbus Driver** as the device type.



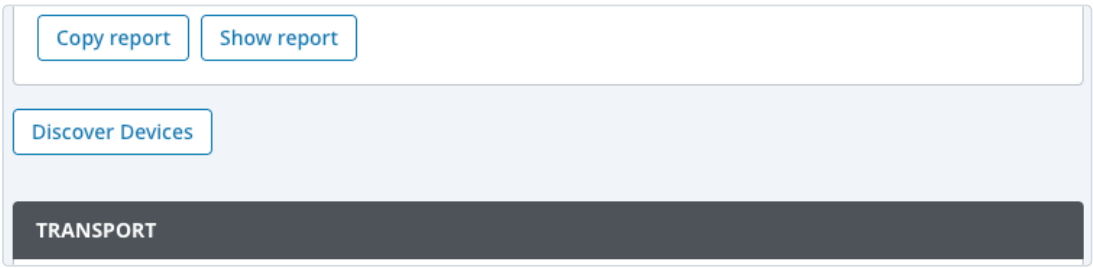
3. Configure the connection settings (see the sections below) and click **Create Device Connection**.

4. The new device then appears in the Device Connections list.

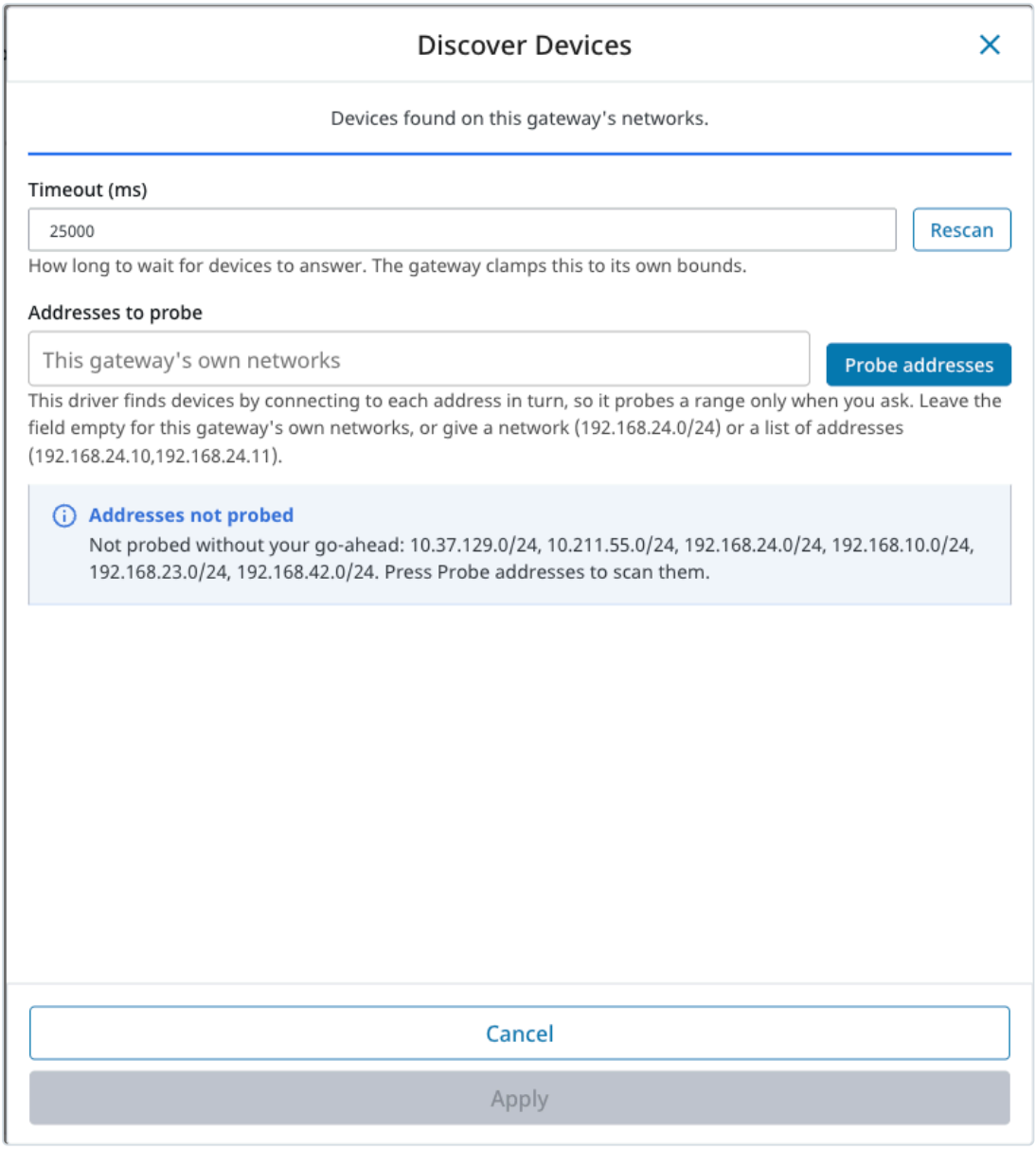


Finding devices on the network

The configuration form carries a **Discover Devices** button. Modbus devices announce nothing, so discovery works differently from the broadcast-based modules: the gateway **connects to each address in turn** on port 502 and asks whatever answers to identify itself. Only a device that answers like a Modbus server is listed; an address with port 502 open that does not speak Modbus is not.



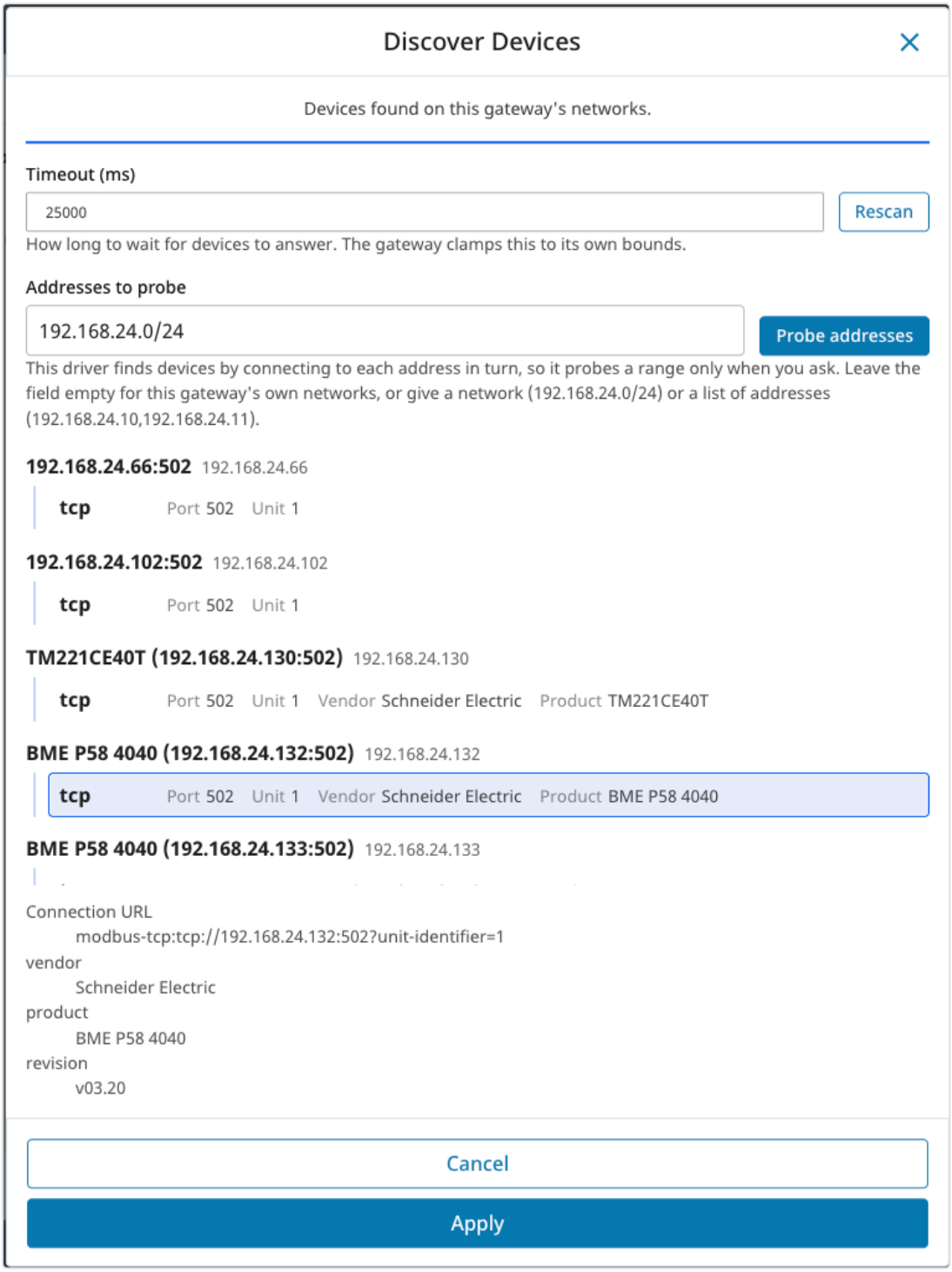
Because that means connecting to every address in a network, **nothing is probed until you say so.** The first sweep after opening the dialog probes no range and reports which networks it left alone, under **Addresses not probed**:



Then:

1. Optionally type what to probe into **Addresses to probe**: a network (`192.168.24.0/24`) or a comma-separated list of addresses (`192.168.24.10,192.168.24.11`). Leave it empty for the gateway's own networks.
2. Press **Probe addresses**. A later **Rescan** probes the same addresses again; closing the dialog forgets the go-ahead.

Selecting a result fills in **Host**, **Port** and **Unit Identifier** — the unit the device answered on, so the connection opens exactly what was probed. Where the device answers the identification request, the result also shows its vendor and product.



- **A range larger than 1024 addresses is refused** before anything is sent, and named under **Range too large**; give a smaller one.
- **The timeout covers the whole probe.** If it runs out, the list shows what answered in time and says so under **Ran out of time**; raise the timeout or narrow the range.
- **Probing crosses routed networks**, unlike a broadcast: a device on another subnet or across a VPN is found as long as the gateway can reach its port 502. A device that serves only a unit other than 1 does not answer the identification request on unit 1; enter it by hand.

- **Probing is visible to the network.** An intrusion-detection system may report it as a port scan; agree it with whoever runs the network before probing a production network.

> There is no *Test Connection* action for this driver: create the device and read its status.

Running in Docker: the discovery agent

A gateway in a Docker container sees only the container's networks, not the plant's: an empty **Addresses to probe** means the container's own network, where there is nothing to find. A range you type works without the agent, because probing crosses routed networks; with the agent, an empty field probes the plant machine's own networks instead. The **ToddySoft discovery agent** closes that gap. It is a small, separate container that runs on the same machine with the host's network and searches the plant networks on the gateway's behalf. The agent only searches: connecting, browsing and reading stay with the gateway, through the machine's normal routing. A gateway installed directly on the plant machine, or a container on the host network, does not need it.

Installing it. The agent comes as an image archive next to this module, in the share you download the module from: `discovery-agent-<version>-<architecture>.tar.gz`, for `amd64` or `arm64`. Use the agent of the module's version. On the machine that runs the Ignition container (Linux):

```
docker load < discovery-agent-1.3.0-amd64.tar.gz
docker run -d --name toddysoft-discovery-agent \
  --restart unless-stopped --network host --cap-add NET_RAW \
  toddysoft/discovery-agent:1.3.0
```

`--network host` lets it search the plant networks, and `NET_RAW` lets it send and capture the discovery frames. Docker Desktop (macOS, Windows) runs containers in a virtual machine, whose networks the agent would search instead of the plant's.

Nothing to set in Ignition. On every **Discover Devices** the module looks for the agent at its container's default gateway, which on Docker is the machine itself. An agent started later is found on the next search, with no restart. A line above the results says what happened:

- *via discovery agent*: the agent searched; its devices are listed together with any the gateway found itself, each once.
- *no discovery agent found · container networks only*: no agent runs; the search is the gateway's own, as before.

- *discovery agent refused: …* or *discovery agent not usable: …*: the reason follows, for example the licence or the shared secret below.

Nothing is probed without your go-ahead, through the agent too: it probes only after **Probe addresses**, and only the addresses typed there or, with the field empty, its machine's own networks.

Licence. The agent searches only for a gateway licensed for this driver. With each search the module sends the licence it runs on, the bundled trial included, over an encrypted connection; the agent checks it, uses it for that search and keeps nothing.

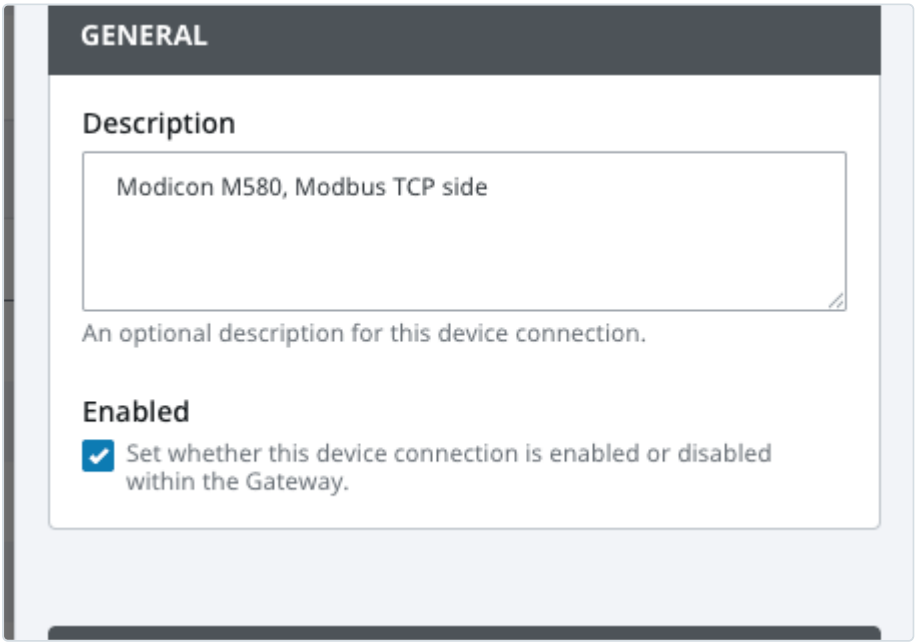
Security. The agent listens only on the Docker networks' own addresses on that machine (port 47900), never on a plant network, and accepts connections only from the container networks. To restrict it further, give it a shared secret: `-e TODDYSOFT_AGENT_SECRET=<secret>` on the agent and `-e TODDYSOFT_DISCOVERY_AGENT_SECRET=<secret>` on the Ignition container. With a secret set, a gateway without it, or with another, is refused. To stop the module looking for an agent, set `TODDYSOFT_DISCOVERY_AGENT=off` on the Ignition container.

Setting blocks

The configuration is organized into categories: **General**, **Transport**, **Modbus**, **TCP** and **Audit Log**.

General

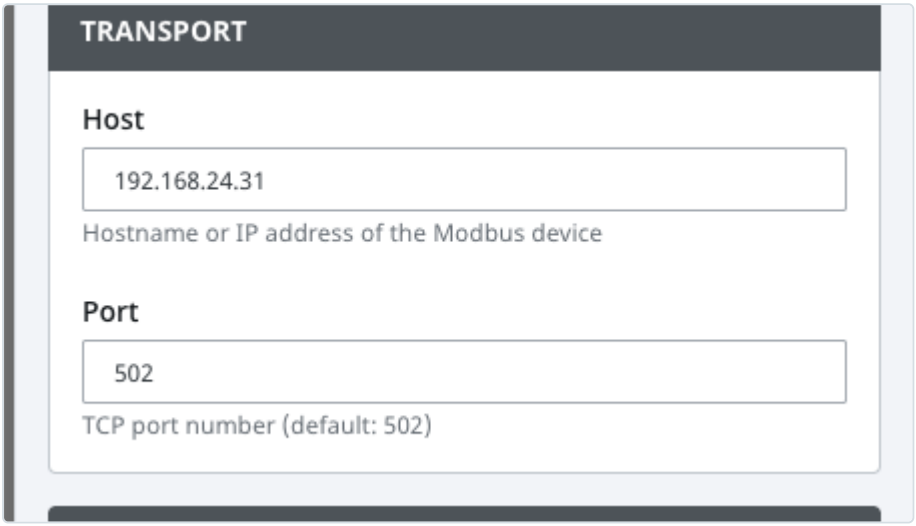
Assigns a name (and optional description) to the connection and controls whether it is enabled.



Setting	Description	Default
Name	Unique name for this device connection	
Description	Optional description for this connection	
Enabled	Whether the connection is active	`true`

This is the name the connection is then referenced under in other parts of Ignition.

Transport



Setting	Description	Default
Host	Hostname or IP address of the Modbus device	
Port	TCP port number	`502`

Modbus

MODBUS

Unit Identifier

1

Unit-identifier or slave-id that identifies the target PLC (default: 1)

Request Timeout

5000

Default timeout for all types of requests in milliseconds

Ping Address

4x00001:BOOL

Address to check connection status (default: holding register 1)

Default Byte Order

BIG_ENDIAN

Default encoding for register values: BIG_ENDIAN, LITTLE_ENDIAN, BIG_ENDIAN_BYTE_SWAP, LITTLE_ENDIAN_BYTE_SWAP

Max Coils Per Request

2000

Maximum number of coils addressable in one request

Max Registers Per Request

125

Maximum number of registers addressable in one request

Max Address Gap To Merge

10

Maximum address gap between tags to merge into single request

Max Concurrent Requests

1

Maximum number of concurrent in-flight requests (increase with caution)

Browse Queries

Query string for symbol browsing. Syntax is protocol-specific; leave empty to disable browsing.

Setting	Description	Default
Unit Identifier	Unit-identifier / slave-id of the target device	`1`
Request Timeout	Milliseconds allowed for one request	`5000`
Ping Address	Address read to check that the connection is still alive	`4x00001:BOOL`
Default Byte Order	Default decoding for multi-register values	`BIG_ENDIAN`
Max Coils Per Request	Most coils addressable in one request	`2000`
Max Registers Per Request	Most registers addressable in one request	`125`
Max Address Gap To Merge	Largest gap between tags still merged into one request	`10`
Max Concurrent Requests	In-flight requests allowed at once	`1`
Browse Queries	Inert on this driver — Modbus cannot enumerate tags	

Default Byte Order

Modbus specifies the order of the bytes inside a register (high byte first) but not the order of the *registers* in a multi-register value, and some devices swap the bytes as well, so vendors genuinely differ and there is no correct default. A wrong setting does not fail — it returns plausible nonsense, which is the failure mode to watch for: a `REAL` that reads as a huge or tiny number, or an integer whose bytes are transposed.

Four values are accepted: `BIG_ENDIAN`, `LITTLE_ENDIAN`, `BIG_ENDIAN_BYTE_SWAP` and `LITTLE_ENDIAN_BYTE_SWAP`. In the terms device manuals use: ABCD is `BIG_ENDIAN`, BADC (byte swap) `BIG_ENDIAN_BYTE_SWAP`, CDAB ("word swapped", "low word first") `LITTLE_ENDIAN_BYTE_SWAP`, and DCBA `LITTLE_ENDIAN`. This is the connection-wide default; individual tags can override it (see *Per-tag options*), which is what you need for a device that is inconsistent with itself.

Request sizing and merging

Max Coils Per Request (`2000`) and **Max Registers Per Request** (`125`) are the protocol's own ceilings. Many devices accept less than the maximum; if a device answers small reads and errors on large ones, lower these before suspecting anything else.

Max Address Gap To Merge lets the driver coalesce nearby tags into one request, reading a few unwanted registers to save a round trip. Raising it trades wire bytes for fewer requests. Beware of merging *across* a gap that the device treats as illegal — some devices fault a read that spans an unmapped address, in which case lower this.

Max Concurrent Requests defaults to `1`, and the description's "increase with caution" is earned: many Modbus devices are single-threaded and answer concurrent requests with errors, dropped frames or transposed responses. Raise it only for a device documented to support it.

Ping Address

The address read periodically to confirm the connection is alive. The default, `4x00001:BOOL`, is holding register 1 — which exists on most devices but not all. If a device faults on the ping while ordinary tags read correctly, point this at a register you know is mapped.

TCP

Socket-level tuning. The defaults are appropriate for nearly every device; change them for a specific diagnosed problem rather than as routine optimisation.

TCP

Connect Timeout

5000

Connection timeout in milliseconds (0 = default)

Read Timeout

0

Socket read timeout in milliseconds (0 = no timeout)

Write Timeout

0

Socket write timeout in milliseconds (0 = no timeout)

TCP No Delay

☑

Enable TCP_NODELAY (disable Nagle's algorithm)

TCP Keep Alive

☐

Enable TCP_KEEPALIVE

Send Buffer Size

81920

Send buffer size in bytes

Receive Buffer Size

81920

Receive buffer size in bytes

Local Address

Local address to bind to (leave empty to bind to all interfaces)

Local Port

0

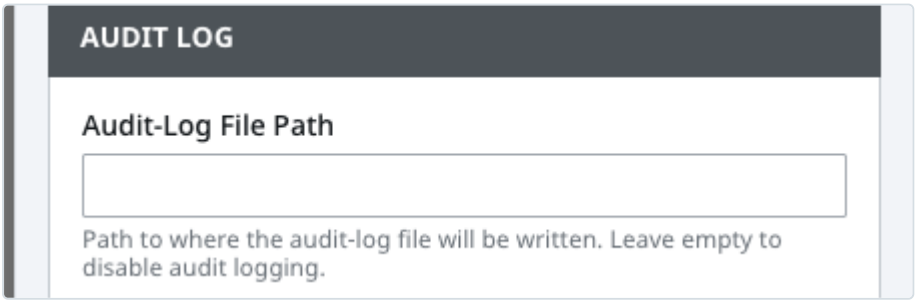
Local port to bind to (0 = use ephemeral port)

Setting	Description	Default
Connect Timeout	Connection timeout in milliseconds (`0` = default)	`5000`
Read Timeout	Socket read timeout in milliseconds (`0` = no timeout)	`0`
Write Timeout	Milliseconds a write may wait for the device to accept data; then the connection is closed and reconnected (`0` = no limit)	`30000`
TCP No Delay	Enable `TCP_NODELAY`, disabling Nagle's algorithm	`true`
TCP Keep Alive	Enable `TCP_KEEPALIVE`	`false`
Send Buffer Size	Send buffer size in bytes	`81920`
Receive Buffer Size	Receive buffer size in bytes	`81920`
Local Address	Local address to bind to; empty binds all interfaces	
Local Port	Local port to bind to (`0` = ephemeral)	`0`

TCP No Delay is on by default because Modbus exchanges are small request/response pairs, exactly the pattern Nagle's algorithm delays.

Local Address is occasionally needed on a multi-homed gateway, to make the connection leave by a particular interface.

Audit Log



Setting	Description	Default
Audit-Log File Path	Path to a file protocol audit events are written to; empty disables	

The audit log records protocol-level events and grows quickly, so enable it only while investigating a problem.

Tag Addressing

```
<area-prefix>:<starting-register>[:<datatype>][[<selection>]]
[{<option>: <value>[, ...]}]
```

Area prefixes are `holding-register`, `input-register`, `coil` and `discrete-input`; Schneider's `%MW0` / `%M0` form is also accepted.

Address	Meaning
`holding-register:1:INT`	One signed 16-bit int, the first holding register
`holding-register:1:DINT`	Signed 32-bit int spanning registers 1–2
`holding-register:1:REAL`	IEEE-754 float spanning registers 1–2
`holding-register:1:STRING(20)`	A 20-character string starting at register 1
`holding-register:1[0..9]:DINT`	Ten 32-bit ints, registers 1–20
`holding-register:1[9]:DINT`	**One** DINT, the tenth
`holding-register:1[0..19]:CHAR`	Twenty individual characters, as a list
`coil:1`	A single coil
`coil:1[0..7]:BOOL`	Eight coils as a BOOL array

> **Register numbers count from 1 within their area.** `holding-register:1` is the first holding register (offset 0 on the wire). A device manual's `40001` is that same register in the classic notation, which the driver also accepts as `4x00001`; written as `holding-register:40001` it is register 40001 instead, usually past the end of the device's map.

> **A bare `[N]` is one element, not a count of N**, and the selection goes **before** the `:TYPE` suffix. Both changed together in ToddySoft Connect 1.2 so that an address written for an earlier release **fails to parse** rather than quietly returning one register where it used to return ten. Each parse error names its replacement.

`STRING` and `WSTRING` must carry a length, because nothing on the wire announces one: `STRING(20)[3]` is three 20-character strings, while `CHAR[20]` is twenty separate characters. A `STRING` without a length is rejected when the address is parsed.

Per-tag options

A trailing `{…}` overrides connection settings for one tag.

Option	Values	Meaning
`unit-id`	0–255	Address this tag to a specific unit / slave
`byte-order`	`BIG_ENDIAN`, `LITTLE_ENDIAN`, `BIG_ENDIAN_BYTE_SWAP`, `LITTLE_ENDIAN_BYTE_SWAP`	Byte order for this tag, overriding the default

```
holding-register:100:INT{unit-id: 5}
holding-register:100[0..9]:DINT{unit-id: 5, byte-order:
LITTLE_ENDIAN}
```

`unit-id` is how you reach several devices behind one connection — a serial multi-drop line behind a Modbus gateway, typically. Tags addressed to different units are never merged into one request, so a device with many units costs at least one request per unit.

Tag Groups

By default, all tags are collected in the default tag group, which means all selected tags are fetched at the default interval of once a second.

If different intervals are desired, create additional tag groups, configure the desired intervals there, and assign tags to the corresponding tag groups.

Modbus has no change notification, so subscriptions are served by polling and the tag group interval *is* the poll rate. Tags that sit close together in the address space and are read at the same interval merge into one request — so grouping by *both* address locality and rate is what keeps the request count down.

Triggered Groups

A triggered group is a set of tags read together, in one request, when a trigger tag changes as a condition describes, instead of at a polling rate: a batch of quality data when a part is finished, for example. Without groups every tag is read at the rate of its tag group.

The **Triggered Groups** card lists the device's groups. **Add group** opens a dialog for one group (name, trigger tag, condition, *Fire on the first value after start* and the members, one address per line); the setting stores them as the JSON list shown below, which can also be pasted in.

```
[
  {
    "name": "part-done",
    "trigger": "<address of the trigger tag>",
    "condition": "oldValue == false && value == true",
    "members": ["<address>", "<address>"]
  }
]
```

Property	Meaning	Default
`name`	1-64 letters, digits, `_` or `-`; unique on the device (case is ignored)	required
`trigger`	The PLC address of the tag whose change fires the group, written as in a tag's address	required
`condition`	When a change fires, over `oldValue` and `value`	`value != oldValue`
`fireOnFirstValue`	Fire once on the first value after the device starts, too	`false`
`members`	The PLC addresses read when the group fires, at most 1,000	none

The controller cannot push changes, so the driver reads the trigger tag at the **Subscription Poll Interval (ms)** (category *Timing*, default 1000 ms). A change that is undone faster than that is not seen; lower the interval for short pulses.

Each group appears in the device's tag tree as the folder `$triggered/<name>`, holding its members, `$trigger` (the trigger's last value), `$fireCount`, `$lastFire` (the time of the last fire), `$coalesced` (fires that came while a read was still running and were folded into the next one) and `$status` (`ESTABLISHED` while it runs, otherwise the state and its reason). A group whose definition is invalid cannot be saved: the device form names every problem. A group keeps its trigger's last value across a reconnect, so a change during an outage still fires once.

Activating the licence

Without a licence the module runs in Ignition's trial: two hours at a time, then its devices stop until someone clicks **Reset Trial** in the gateway's banner, as often as needed. A licence for this module comes from ToddySoft by email as an **Ignition licence key** with an **activation token**. The ToddySoft licence for the drivers travels inside that key: activating it on the gateway is all there is to do. There is no licence file to copy and no licence setting in the device form. Keep the key and the token private: together they activate the licence.

Online (the gateway can reach the internet)

1. Install the module first (see Installation).
2. In the gateway's web interface open **Platform > System > Licensing**, click **Activate a License**, enter the **License Key** and click **Next**.
3. Choose **Online Activation** and click **Activate License**. If the gateway asks for an activation token, paste the token from the email: one line, without the `BEGIN` and `END` lines.
4. The licensing page shows the key as activated, and **Platform > System > Modules** shows this module as **Activated**.

The gateway asks Inductive Automation's activation service (`api.inductiveautomation.com`, HTTPS) for this; a firewall must allow that one outgoing connection.

Offline (the gateway cannot reach the internet)

Offline activation is a round trip with two files, between the gateway and any computer with internet access. It works for six-character licence keys.

1. On the gateway open **Platform > System > Licensing**, click **Activate a License**, enter the **License Key** and click **Next**.
2. Choose **Offline Activation**, then click **Download Activation Request**. If the gateway asks for an activation token, paste the token from the email. The gateway saves the file `activation_message.txt`.
3. On a computer with internet access open https://links.inductiveautomation.com/activation, choose `activation_message.txt` and save the `license.ipl` file it returns.
4. Back on the gateway's **Offline Activation** page, upload `license.ipl` and click **Activate**.

After activating

No restart is needed. The module takes the licence as soon as Ignition reports the activation, and devices that were refused for want of a licence connect again on their next attempt. The licence status shows **VALID** with the tier and the end of maintenance (see Licence diagnosis); anything else is explained there.

Renewing or extending a licence

- **When ToddySoft has updated the licence on your existing key** (more drivers, longer maintenance), the gateway has to fetch it again. Online: in **Platform > System > Licensing** open the key's three-dot menu and choose **Reactivate**. Offline: unactivate the key first (below), then activate it offline again. Inductive Automation warns that an offline activation without unactivating first can put the gateway into emergency activation mode.
- **When you received a new key**, activate it and unactivate the old one. Keep exactly **one** activated key covering this module per gateway: with two, Ignition hands the module only one of them (see Troubleshooting).

Moving the licence to another gateway

Unactivate the key on the old gateway, then activate it on the new one. The email says how many gateways a key may be activated on at the same time.

- Online: in **Platform > System > Licensing** open the key's three-dot menu, choose **Unactivate** and confirm with **Unactivate**.
- Offline: in **Platform > System > Licensing** open the key's three-dot menu, choose **Unactivate** and confirm with **Unactivate**. The gateway saves `unactivation_message.txt`; upload it on https://links.inductiveautomation.com/activation from a computer with internet access.

Ignition Edge: a licence for a third-party module on Ignition Edge is arranged through Inductive Automation. Contact ToddySoft before you order one.

Licence diagnosis

When a device reports a licence problem, the module explains it on the gateway itself. Open the device's settings (**Connections > Devices > Connections**, edit the device, or create a new one): the **Licence** card at the top of the form shows the diagnosis. It is on the form for a new device too, so the licence can be checked before the first device exists.

The diagnosis shows:

- **which gateway** this is, and its role if it is part of a redundant pair;

- **Ignition's licence** for this module on this gateway: activated, trial running (with its end) or trial expired;
- **what ToddySoft licence data arrived** with Ignition's licence: how many parts, whether they are complete, and a checksum that identifies the licence without revealing it;
- the **driver's own verdict** and the **cause**, in words.

Cause	Meaning	What to do
`IGNITION_UNLICENSED`	Ignition applies no activated licence to this module on this gateway (trial running or expired). ToddySoft licence data only arrives with an activated licence.	Activate a licence key that covers this module on this gateway. In a redundant pair every node needs its own activation. If a key was activated and this cause remains, see Troubleshooting: an older, expired key covering this module may be the one Ignition judges it by.
`NO_DATA_FOR_DRIVER`	The activated key carries no ToddySoft licence for this driver.	Send the report to ToddySoft.
`DATA_INCOMPLETE`	The licence data arrived incomplete or damaged.	Send the report to ToddySoft; the licence has to be delivered again.
`DATA_REJECTED`	The driver refused the delivered licence; the reason is shown.	Send the report to ToddySoft.
`RUNNING_ON_BUNDLED_TRIAL`	A licence was delivered but not accepted, and the driver runs on its built-in trial until that ends.	Send the report to ToddySoft.
`NOT_CHECKED`	Licence data arrived, but the driver gave no verdict - for instance because it could not be loaded.	Send the report to ToddySoft.

Cause	Meaning	What to do
`VALID`	Licensed; tier and expiry are shown.	Nothing.
`NOT_YET_KNOWN`	The first seconds after start-up, before Ignition reports a licence state.	Wait a moment.

Sending a report to support. The diagnosis comes as a plain-text report, the one thing to send with any licence question. **Copy report** puts it on the clipboard; **Show report** displays it. The same text is served at `http://<gateway>:<port>/data/com.toddysoft.connect.modbus/licence-report` to a logged-in user. It contains no licence data, no keys, no credentials and no customer details. In a redundant pair, send the report from both nodes.

The gateway log carries the same report once each time the licence situation changes.

Troubleshooting

Licensing problems after a new or renewed licence key was activated — `DATA_INCOMPLETE` with "duplicate fragment index", or `IGNITION_UNLICENSED` with "trial expired", in the licence diagnosis:

- **Two licence keys covering this module are activated on the gateway at the same time**, typically a renewed or upgraded key activated alongside the old one. Ignition hands the module the licence data of both, in numbered parts that carry the same numbers; the module cannot tell which parts belong together, keeps one of each pair, and the licence it reassembles is unreadable. Depending on the parts it keeps, devices are refused or run on the module's bundled trial until that ends.
- If one of the two keys has **expired**, Ignition may judge the module by that key alone: the diagnosis shows `IGNITION_UNLICENSED` ("trial expired - no activated licence applies to this module") and "Data: none received", although the new key was activated successfully.
- Keep exactly **one** activated licence key that covers this module per gateway: deactivate the old one under **Platform > System > Licensing**, then restart the module (or the gateway). In a redundant pair, check every node.

The module will not load, or the log shows `ClassNotFoundException` / `NoClassDefFoundError`:

- Ignition's **OPC UA** module is missing or disabled. This driver is an OPC UA device driver and cannot load without it. Install or re-enable it under **Platform > System > Modules** and restart the gateway — see Installation

Device shows "Faulted":

- Verify the device is reachable and TCP port 502 (or your override) is open
- Check the **Unit Identifier** — a device behind a gateway rarely answers on the default `1`
- Check the **Ping Address** exists on this device; the default is holding register 1
- Review Gateway logs: **Diagnostics > Logs**

Values are plausible but wrong — floats absurd, integers transposed:

- Almost always **Default Byte Order**. Try the other three values; override per tag with `{byte-order: …}` for a device that is inconsistent with itself

Small reads work, larger ones fail:

- Lower **Max Registers Per Request** / **Max Coils Per Request**. The protocol ceilings are not what every device accepts
- Or lower **Max Address Gap To Merge**: a merged read may be spanning an unmapped address the device refuses

Intermittent errors, dropped or mismatched responses:

- Set **Max Concurrent Requests** back to `1`. Many devices are single-threaded

Nothing to browse in the Designer:

- Expected. Modbus has no tag enumeration; write addresses by hand from the device's Modbus map

An address that used to work no longer parses:

- The array grammar changed: the selection goes before the type, and `[N]` is one element rather than a count. The error message names the replacement

Docker related issues

The Modbus protocol carries no identity for this end of the connection, so a container's internal address is never announced to the device and nothing has to be reconciled. A gateway in a bridged container connects normally, provided the device's port is routable.

Device discovery needs the discovery agent. *Discover Devices* searches the networks the gateway itself is attached to, which for a container are Docker's, not the plant's. Run the ToddySoft discovery agent next to the container — see Running in Docker: the discovery agent — or enter the host by hand.

VPN related issues

A remote device reached over a VPN needs nothing special from this driver: the connection is an ordinary outbound TCP session. Note that latency matters more here than on most drivers, because Modbus is strictly request/response — with **Max Concurrent Requests** at `1`, every tag read costs a full round trip.

Version history

1.3.0 — 2026-11-13

- **New:** triggered groups - tags read together when a trigger tag changes, instead of at a polling rate. See Triggered Groups.
- **Fixed:** writes to extended registers (`6x`) work; every one failed before it was sent. Large merged writes stay within Modbus's limit per request.
- **Fixed: Write Timeout** is back and takes effect, with a default of 30 s. If the device stops accepting data, a write now fails after this time and the connection is reconnected, instead of hanging for ever. Earlier versions offered the setting but never applied it, so a value saved on an existing device is not taken over; set it again if you need a value other than 30 s.
- **Fixed:** the installation steps for Ignition 8.3: the gateway must be restarted after installing or upgrading the module, the menu is **Platform > System > Modules** and the status reads **ACTIVE**. All menu paths follow Ignition 8.3 now (devices under **Connections > Devices > Connections**, logs under **Diagnostics > Logs**). See Installation.
- **New:** how to activate the licence key, online and offline, renew it and move it to another gateway. See Activating the licence.
- First release of this manual.
- **New: Discover Devices** finds Modbus/TCP devices by probing addresses, with your go-ahead. See Finding devices on the network.
- **New:** licence diagnosis — why a device is or is not licensed, shown on the gateway, with a report to send to support. See Licence diagnosis.
- **New:** troubleshooting for two licence keys covering this module activated on one gateway at the same time. See Troubleshooting.
- **New:** discovery when Ignition runs in Docker, through the ToddySoft discovery agent. See Running in Docker: the discovery agent.
- **Fixed:** a write the PLC refuses now fails in Ignition, with the reason as the status (for example `Bad_TypeMismatch` for a value of the wrong type). Until now every write was reported as successful, and the refusal appeared only in the gateway log.
- **Fixed:** a tag that has not been read yet, or whose last read failed, no longer reads 0 with Good quality. Before its first read it has no value and the quality Uncertain; after a failed read it keeps its last value with Bad quality until the next good read.

- **Fixed:** removing the last tag of a tag group (or polling rate) no longer leaves the device showing *Connection Lost*. A poll still running at that moment made the device fail with `Batch '...' not found` in the gateway log and stay in that state until it was restarted.