ToddySoft Connect EtherNet/IP Driver - Ignition 8.3

Ignition 8.3 module for connecting to Allen-Bradley **Logix** controllers — ControlLogix, CompactLogix and family — over EtherNet/IP and CIP. Tags are addressed by the **symbol names from the Logix program**, controller-scoped or program-scoped, with dotted paths for struct members and bracketed indices for arrays.

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.

> **Reads are batched, writes are per-tag.** The driver coalesces a multi-tag read into one CIP multi-service request, so a tag group of many tags costs far less than the tag count suggests. Aggregates are a deliberate exception in the other direction — see *Arrays and structures*.

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-EIP-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.

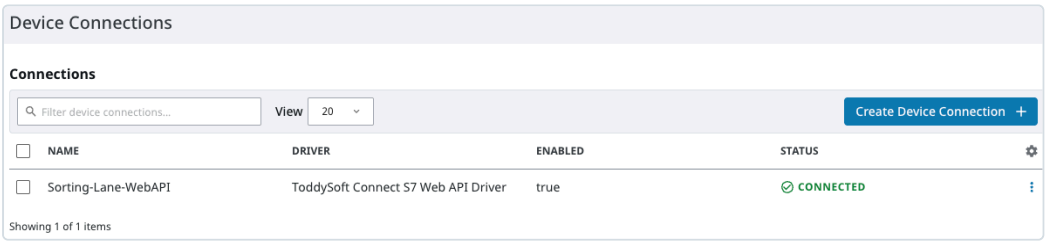
Before you start: the controller side

- **EtherNet/IP enabled.** CompactLogix and ControlLogix CPUs with built-in Ethernet have it on by default; a ControlLogix with a separate 1756-ENBT / 1756-EN2T card needs that card configured.
- **"Allow external access" enabled** in the program properties (Studio 5000 / RSLogix 5000). Some sites disable it by default, and with it disabled the connection is made and the tags are not readable.
- **The tag database must be accessible.** An empty or password-locked program database prevents symbolic browsing — there is nothing else to address by.
- **TCP port 44818** reachable from the gateway host.

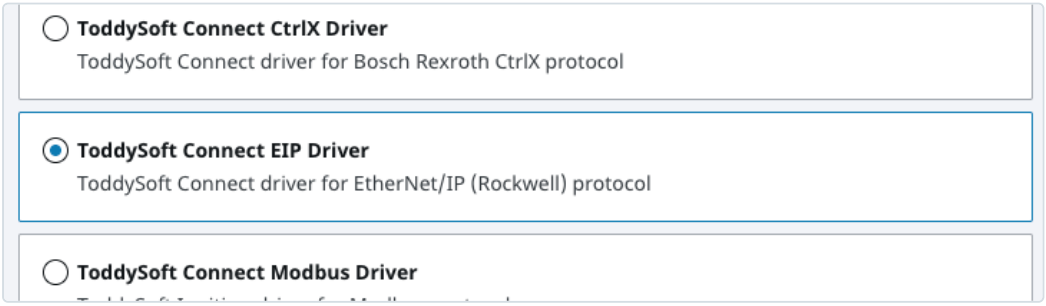
There are no credentials to configure: EtherNet/IP as this driver speaks it is unauthenticated, so this module has no username or password fields at all. Access is governed on the controller.

Creating a Device Connection

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

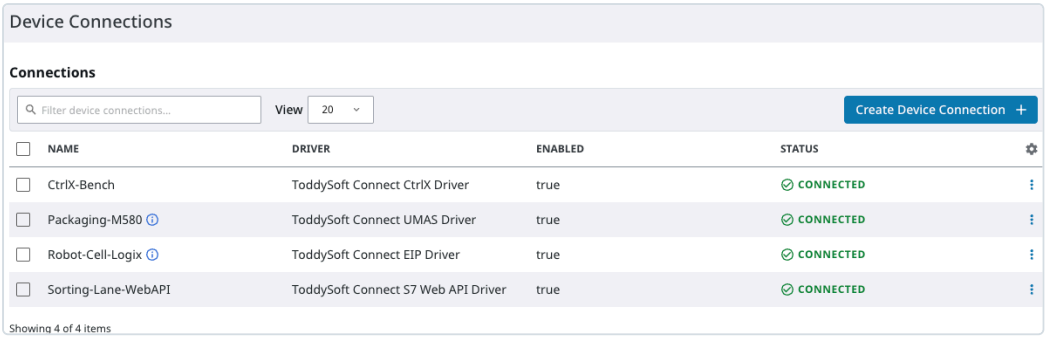


2. Click **Create Device Connection** and select **ToddySoft Connect EIP Driver** as the device type.



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.



> There is no *Test Connection* action for this driver on either Ignition version — create the device and read its status.

Finding devices on the network

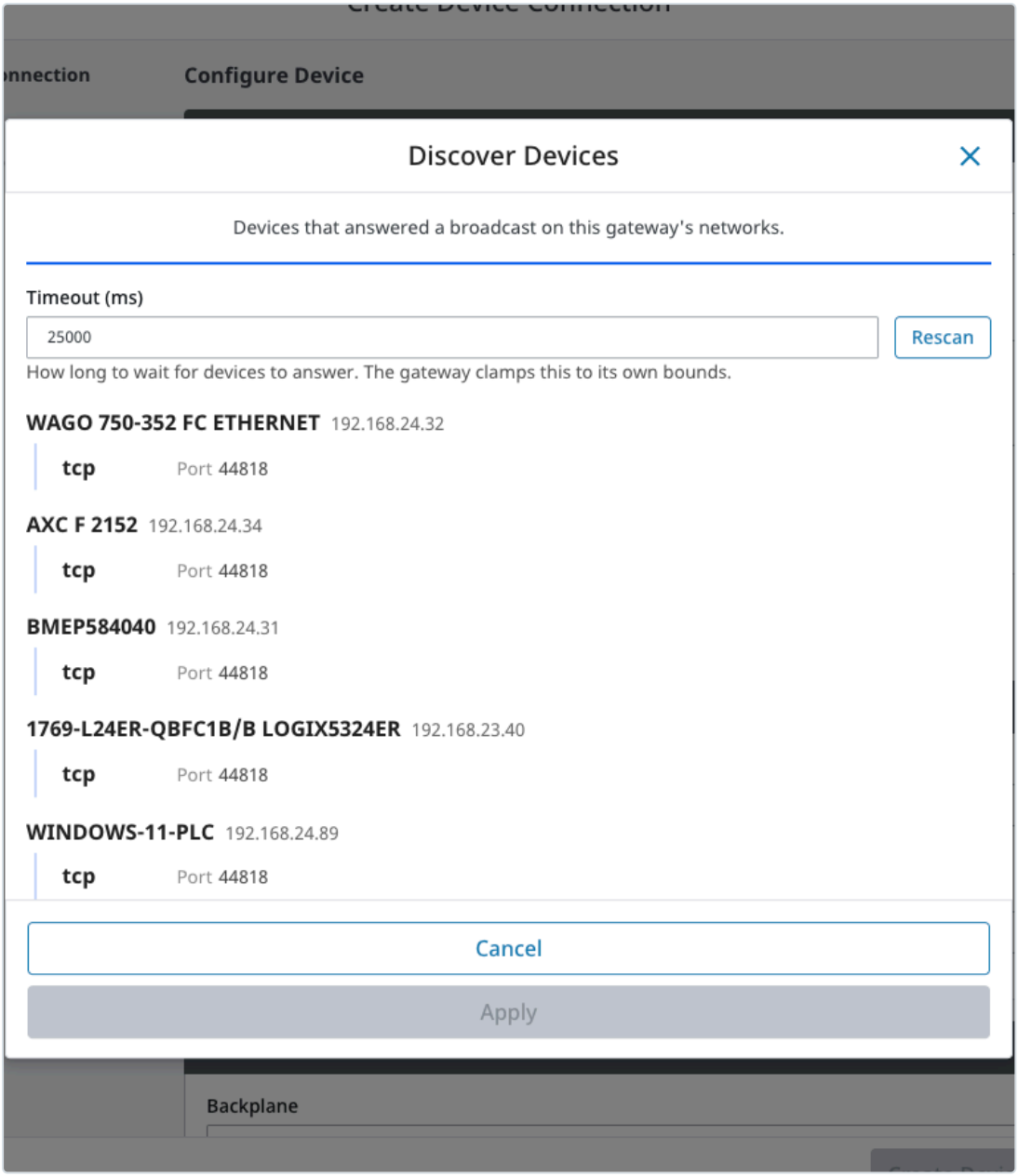
The configuration form carries a **Discover Devices** button, which sweeps the gateway's own networks using EtherNet/IP's **LIST IDENTITY** broadcast and lists the devices that answer.



Selecting a result fills in **Host** and **Port**. Two things it deliberately does not do:

- **It does not fill in Slot or Backplane.** LIST IDENTITY says nothing about where in the chassis the controller sits, so those are left at their defaults rather than guessed at. Check them against the chassis layout — a wrong pair is the commonest cause of a rejected ForwardOpen.
- **It fills in only what the device actually reported.** A field the sweep said nothing about is left alone.

A broadcast does not cross a routed network. A controller on another subnet, or at the far end of a VPN, will not appear however long you wait; enter its address by hand.



Running in Docker: the discovery agent

A gateway in a Docker container sees only the container's networks, not the plant's: the LIST IDENTITY broadcast behind *Discover Devices* never leaves the container's network, so no device answers. 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.

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.

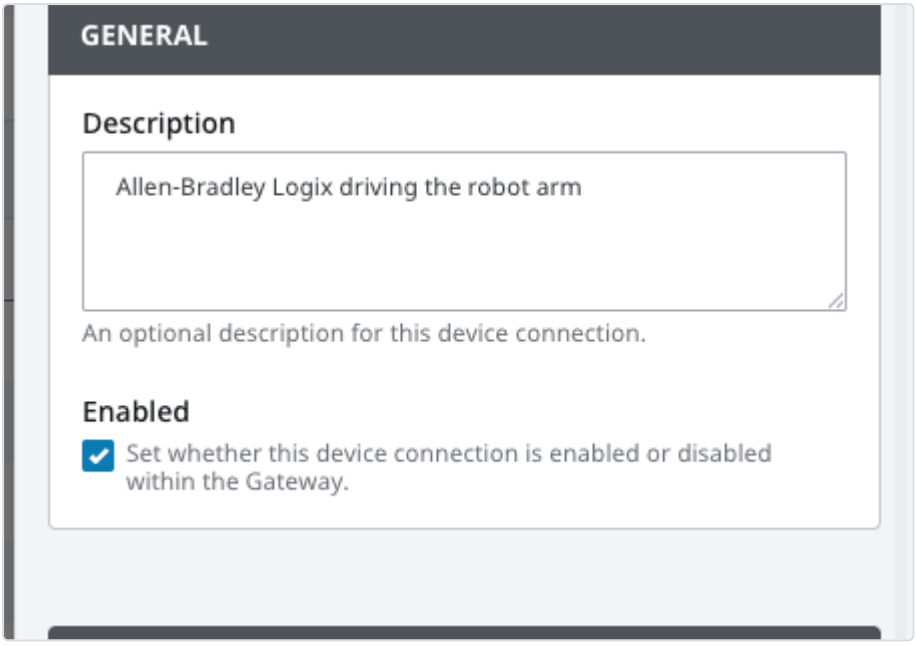
Security. The agent listens only on the Docker networks' own addresses on that machine (port 47900), never on a plant network, and accepts connections only from the container networks. To restrict it further, give it a shared secret: `-e TODDYSOFT_AGENT_SECRET=<secret>` on the agent and `-e TODDYSOFT_DISCOVERY_AGENT_SECRET=<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**, **EIP**, **Limits** and **Audit Log**.

General

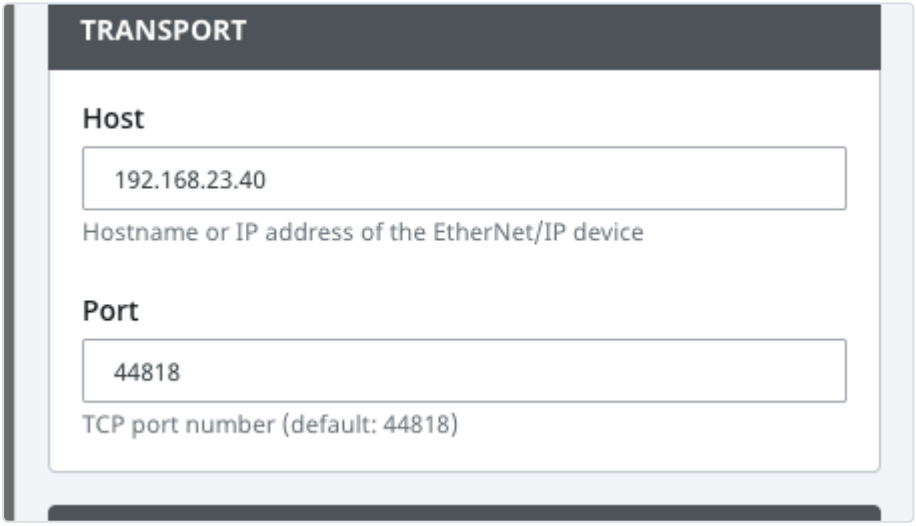
Assigns a name (and optional description) to the connection and controls whether it is enabled.



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.

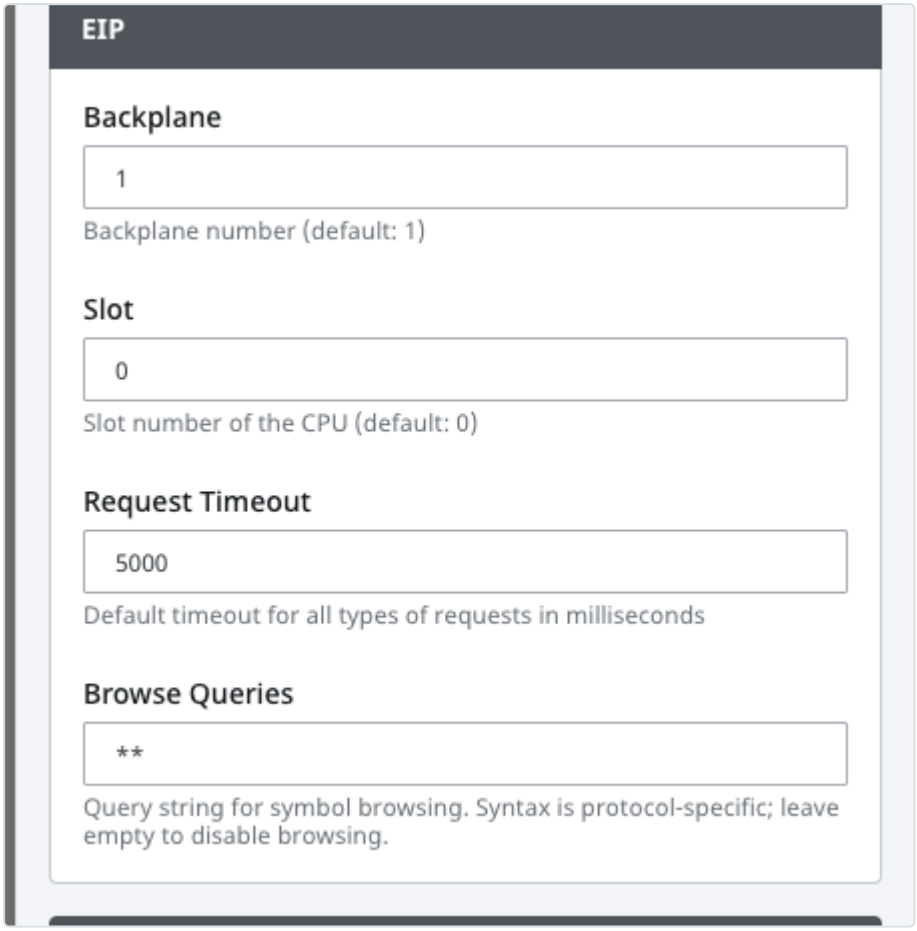
Transport



Setting	Description	Default
Host	Hostname or IP address of the EtherNet/IP device	
Port	TCP port number	`44818`

There is no transport selector, and none is missing: EtherNet/IP explicit messaging runs over plain TCP on 44818, and there is no TLS variant to choose here.

EIP



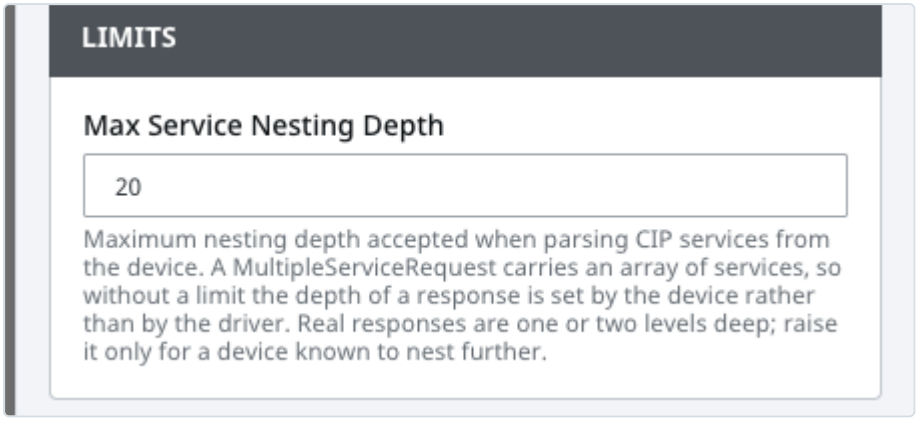
Setting	Description	Default
Backplane	Backplane number of the target controller	`1`
Slot	Slot number of the target controller	`0`
Request Timeout	Milliseconds allowed for one request	`5000`
Browse Queries	Query string for symbol browsing	`**`

Backplane and slot are worth double-checking

They identify *which module in the chassis* the CIP route ends at. A wrong pair does not usually fail politely: a **ForwardOpen rejected with CIP general status `0x01`** is the common symptom, and the same status is what you get when the controller simply has **no free connection slots** — an open Studio 5000 session counts against that budget. Check the chassis layout first, then look for other clients.

For a CompactLogix with a built-in Ethernet port the controller is usually backplane `1`, slot `0`, which is the default here.

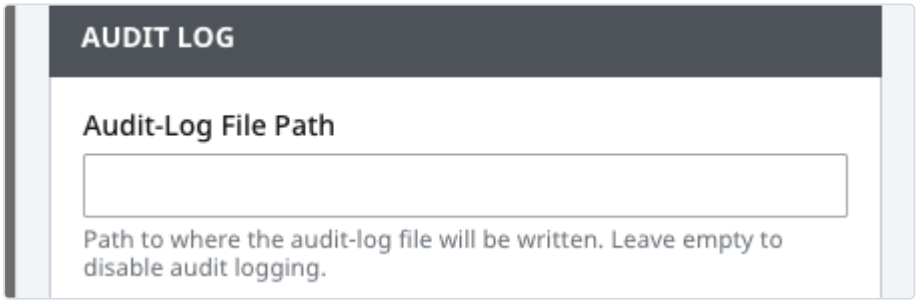
Limits



Setting	Description	Default
Max Service Nesting Depth	Nesting accepted when parsing CIP services from the device	`20`

This bounds what the *device* can make the driver parse. A CIP `MultipleServiceRequest` carries an array of services, so without a limit the nesting depth of a response is chosen by the device rather than by the driver. Real responses are one or two levels deep; the default of `20` sits far above that. Raise it only for a device known to nest further.

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

EIP addresses are **symbolic** — the names from the Logix program:

```
<scope>:<tag>(.<member>)*([<index>(,<index>)*])*
scope ::= (omitted = controller-scoped) | <program-name>
```

Address	Meaning
`MyTag`	A controller-scoped tag
`Program:MainProgram.LocalTag`	A program-scoped tag inside `MainProgram`
`MyStruct.Field`	A struct member
`MyArray[5]`	Element 5 of an array
`MyArray[2,3]`	A cell of a multi-dimensional array
`Channels[0].Id`	A selection on an intermediate path segment
`Matrix[0].Row[1]`	Selections on more than one segment

Controller-scoped versus program-scoped is the usual cause of `INVALID_ADDRESS`. A tag local to a program needs the `Program:<name>.` prefix; a controller-scoped tag must not have one. Browse to discover the legal names rather than transcribing them from Studio 5000.

An index above 255 is refused, because a CIP `MemberID` cannot encode it. A range before the last path segment is refused too, because one request carries a single element count.

Arrays and structures

Naming an array without a subscript reads the whole array. A 3×4 reads as 12 elements in wire order, and the count comes from the controller's tag table, so it agrees with what browsing reports.

- **Packed BOOL arrays are unpacked for you.** Logix stores `BOOL[32]` as one `DWORD`; the driver returns 32 `BOOL`s, least significant bit first. Reading a single element such as `g_arrBool[3]` selects the bit. **Writing** a single element is refused, because changing one bit of a shared word needs a read-modify-write that would clobber the other 31 if anything else touched them meanwhile. Write the whole array instead.
- **Strings read whole**, including user-defined string types (`STRING_16`, `STRING_32`, …). A string is recognised by its layout — a `DINT LEN` followed by a `SINT DATA` array — not by one hardcoded type.

- **An array of UDTs does not read.** The driver decodes no other user-defined structure, so such a tag reports `UNSUPPORTED` rather than a partial value.
- **A member array resolves through the controller's struct definitions**, one-dimensional members only: the type information a controller publishes carries an element count for those alone, so a multi-dimensional member reads one element and should be addressed per element.

An aggregate is readable but not writable, and browsing says so — a structure or array node reports `readable=true`, `writable=false`. The asymmetry is deliberate: the driver decodes a structure on read through the controller's own template definitions, and has no encoding for one on write.

Configure **Browse Queries** to control which symbols appear when browsing tags in the Ignition Designer.

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.

Reads within one group are coalesced into a CIP multi-service request, so grouping tags that are read together is what makes this driver efficient — spreading the same tags across many groups costs more, not less.

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*, trigger poll rate 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
`pollRateMs`	How often the trigger tag is read, in ms (at least 10)	`100`

EtherNet/IP cannot report changes, so each group reads its trigger tag every `pollRateMs` milliseconds (*Trigger poll rate* in the dialog), 100 ms when not set, at least 10 ms. A change that is undone between two reads is not seen.

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.
`VALID`	Licensed; tier and expiry are shown.	Nothing.

Cause	Meaning	What to do
`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.eip/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 controller is reachable and TCP port 44818 is open through any site firewall
- Confirm EtherNet/IP is enabled on the controller or its communications card
- Review Gateway logs: **Diagnostics > Logs**

ForwardOpen rejected, CIP general status `0x01`:

- Wrong **Backplane** / **Slot** for the chassis layout, **or** the controller has no free connection slots. An open Studio 5000 session counts against that budget — close other clients and retry

ForwardOpen rejected, CIP general status `0x09`:

- The routing path does not match a real route in the chassis. Re-check the chassis hardware

`INVALID_ADDRESS` on a symbolic read:

- The tag does not exist in the cached symbol table, or the scope is wrong. `Program:Name.Tag` for program-scoped, bare `Tag` for controller-scoped

`OUT_OF_RANGE` on an array read:

- The index is past the array's declared bounds. Check the dimensions in Studio 5000 or by browsing

CIP general status `0x05` (Object does not exist):

- The tag was deleted, or the program was redownloaded with a different schema. Re-browse; the cache invalidates on a tag-list version change, but a stale cache may need an explicit reconnect

CIP general status `0x10` (Device state conflict):

- The controller is in a mode that disallows the operation. Writes generally need RUN mode

Tags read but nothing can be browsed:

- "Allow external access" is disabled in the program properties, or the tag database is empty or password-locked

A write to an array or structure is refused:

- Expected, and deliberate. Aggregates are read-only on this driver; write the individual elements, or the whole array in the packed-BOOL case

Docker related issues

The EtherNet/IP protocol carries no identity for this end of the connection, so a container's internal address is never announced to the controller and nothing has to be reconciled. A gateway in a bridged container connects normally, provided port 44818 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 controller reached over a VPN needs nothing special from this driver, for the same reason: the connection is an ordinary outbound TCP session. Note that **Discover Devices** uses a LIST IDENTITY *broadcast*, which will not cross a routed VPN — enter such a controller's address by hand.

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.
- **Changed: Backplane** takes effect when the connection is opened; ToddySoft Connect ignored it before. A backplane and slot that lead to no controller now fail the connect with an error, instead of the gateway talking to the Ethernet module itself. Check both settings after upgrading.
- **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:** 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.
- **Changed:** date and time tags have new value types. DATE and DATE_AND_TIME are DateTime values (until now a DateTime tag holding epoch milliseconds as a

number); TIME and LTIME are milliseconds, TIME_OF_DAY and LTIME_OF_DAY milliseconds since midnight (Int64). Writes take the same forms. Adjust scripts and bindings that used the old values.

- **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.
- **Changed:** a browsed array is one folder, named after the array and its index range (for example `arr[0..9]`), holding its elements; until now the elements sat directly beside the array's neighbours. A multi-dimensional array, until now left out of the browse, appears as nested folders down to its elements (`cube[1..2][1..2]`, then `cube[1][1..2]`, then `cube[1][1]`). An array of structures shows a folder per element holding its members (for example `pts[2].x`). Each element keeps its OPC item path, so existing tags keep working.