Amtiri

REFERENCE MANUAL

# EtherNet/IP

Generated on November 25, 2026

The `EIP-Client` module reads and writes the tags of a Rockwell Logix controller **by name**, over EtherNet/IP explicit messaging on TCP port 44818.

| Module | What it does | Typical use |
|---|---|---|
| `EIP-Client` (EtherNet/IP Client) | Reads controller tags into variables on a schedule, and writes variables to tags when they change | A ControlLogix or CompactLogix controller, read and written by tag name |

**Enable**, **Name** and the **Status and messages** tab work the same way in every module. They are described in Settings every module shares.

> **Tip**
>
> On a controller whose firmware carries an OPC UA server, prefer the OPC UA client: it reaches program-scoped tags, which this module cannot, and it authenticates and encrypts.

## EtherNet/IP Client

The module has the two lists every client module has: **collectors** read a tag into a variable on a schedule, and **forwarders** write a variable to a tag whenever the variable changes.

## Settings

| Key | Setting | Type | Default | Meaning |
|---|---|---|---|---|
| `host` | Host | string | (empty) | The controller's address. Required: without it the module does not open |
| `port` | Port | int | 44818 | The TCP port |
| `communicationPath` | Communication path | string | (empty) | The CIP route to the CPU when it is not the device answering at **Host**, such as `1,0`: the backplane, slot 0. Empty sends no route |
| `requestTimeoutMs` | Timeout (ms) | int, ms | 5000 | How long one request may take before it fails |
| `bigEndian` | Big-endian | boolean | false | Sends the encapsulation header big-endian. Off is what the EtherNet/IP specification says; turn it on only for a controller that answers nothing otherwise |
| `forceUnconnectedOperation` | Unconnected requests | boolean | false | Sends unconnected requests, without the CIP Connection Manager. For a gateway or a simulator that does not implement it and closes the session when asked to connect |
| `collectors` | Collector | list of rows | empty | The tags read into variables |
| `forewarders` | Forwarder | list of rows | empty | The variables written to tags. The key is spelled `forewarders` |

The editor's note beside the two checkboxes says it plainly: change them only if the controller does not answer. A ControlLogix normally wants neither.

## Collectors and forwarders

A collector row uses every key below. A forwarder row uses `field` and `tag`.

| Key | Column | Type | Default | Meaning |
|---|---|---|---|---|
| `field` | Variable | variable code | (none) | Collector: the variable the tag's value is written to. Forwarder: the variable whose changes are written to the tag |
| `tag` | Tag | string | (empty) | The tag's name and type, written as the next section says |
| `cron` | CRON | Quartz cron | `0 * * * * ?` | When the collector is read. Required for a collector |

The **Type / problem** column beside each tag is not stored. It shows what is wrong with the tag, or, when nothing is, the variable type the tag yields.

## Naming a tag

A tag is **a name, then a colon and a type**, with an optional element or slice before the colon:

```
Filter1_DP:REAL          a controller-scoped tag
Backwash.Active:BOOL     a member of a UDT instance, at any depth
Counter[3]:DINT          one element of an array
Counter[0..9]:DINT       ten elements, which arrive as a list
```

- **The type is required, and written in capitals.** Without it the driver reads every tag as a `DINT`, so a REAL tag written `Filter1_DP` would return a plausible, wrong number. The module refuses a tag without a type instead.
- **Names** start with a letter or `_`, and hold letters, digits and `_`. Members of a structure are joined with `.`. Each name is at most 40 characters, the Logix limit.
- **An element `[i]` or a slice `[a..b]`** comes last, just before the colon: `Tanks[3].Level:REAL` cannot be written. An index runs from 0 to 255, a slice may not end before it starts, and an array has one dimension: `Counter[256]:DINT` and `Counter[0..9,0..3]:DINT` are both refused.
- **There is no browsing.** The driver cannot read a controller's tag list: tags are typed, or imported from Studio 5000 (below).

> **Important**
>
> **Program-scoped tags cannot be read.** `Program:MainProgram.Tag` has no form the driver can express, so the module refuses any tag that starts with `Program:` or holds more than one colon. Move the tag to controller scope, or reach it through the OPC UA client.

| Type | Variable type |
|---|---|
| `BOOL` | boolean |
| `SINT`, `USINT`, `BYTE`, `INT`, `UINT`, `WORD`, `DINT`, `UDINT`, `DWORD`, `LINT`, `ULINT`, `LWORD` | int |
| `REAL`, `LREAL` | float |
| `STRING` | string |

A slice yields a list of its type. The variable type is what the **Type / problem** column shows, and what the pencil beside **Variable** creates a variable as. A value written is converted to the width of the tag's type first, and an unsigned type travels in the next wider signed type, so a `UDINT` of 3000000000 arrives whole. Keep a forwarded variable within its type's range: the module does not check it.

What the **Type / problem** column, and the error of a module that will not open, say about a tag:

| Text | Meaning |
|---|---|
| `No tag` | The tag is empty |
| `'{tag}' is a program-scoped tag, which this module cannot read; move it to controller scope` | It starts with `Program:`, or holds more than one colon |
| `'{tag}' names no type; write it as {tag}:REAL, with the type the controller holds` | No colon and type |
| `'{tag}' is not a tag address this module understands` | It does not follow the rules above, a lower-case type included |
| `'{type}' is not a type this module understands` | The type is not in the table |
| `'{name}' is longer than the 40 characters a Logix name may have` | A name or member over 40 characters |
| `'{tag}' ends before it starts` | A slice whose end is before its start |
| `'{tag}' asks for index {n}; a CIP request reaches 0 to 255` | An index over 255 |

## Importing tags

Export the tag list from Studio 5000 with **Tools # Export # Tags**, then press **Import tags** and choose the file. The import reads the export's comma-separated rows, in UTF-8, up to 4 MB:

- **Controller-scoped tags of a type in the table become collectors**, on the schedule `0 * * * * ?`.
- **An array becomes its first element.** A tag exported as `REAL[10]` becomes `Name[0]:REAL`: how much to read at once is a decision with a cost, and the table is where it is made.
- **Everything else is skipped and counted**: program-scoped tags, types the module does not carry (every UDT included, whose members have to be named one by one), and names already in the table, compared without regard to case, so importing the same file twice does not double it.

The result is shown where **Test read** shows its answer, in the form `{n} tag(s) imported. Skipped: {n} program-scoped, {n} of an unsupported type, {n} already in the table.`

An imported row has no variable yet. Choose one for each row, or create it with the pencil, before enabling the module.

### Test read

Type a tag in the box beside **Test read** and press the button. The engine reads that tag once, with the **Host**, **Port**, **Communication path**, **Big-endian** and **Unconnected requests** on the screen and a timeout of 5000 ms, and shows the value, or why it could not read it: the tag's problem, `No host`, `The controller answered {code}`, or the driver's reason. Nothing is saved, and the tag does not have to be in a table.

### Behaviour

- **Opening.** The host, every collector's schedule and every tag, collectors and forwarders alike, are checked before anything starts. A problem stops the module opening, naming the variable of the row.
- **Collectors sharing a schedule are one request.** Collectors whose cron expressions are written exactly the same are read together: a hundred tags on one schedule cost one round trip, not a hundred.
- **Every collector is read once when the module opens**, then on its schedule. The first read opens the connection: a controller that is switched off does not stop the module opening, it shows as a lost connection.
- **There are no subscriptions.** Every collector is polled.
- **A refused tag is not a lost link.** When the controller refuses one tag, its variable is left as it was, the rest of the request is written, and the module warns.
- **Reconnection is the module's own.** A request that fails, or takes longer than **Timeout (ms)**, drops the connection, and the next read or write opens a new one. The schedules pace the retries.
- **Forwarders** write their variable's current value once when the module opens, then every change, one request per write.

### Messages

A problem is written into the module's message variables, and the next successful read clears it. A refused tag is the exception. A read in which the controller refuses one tag still counts as a successful read, and every other tag of it is written, but the warning about the refused tag stays while the controller keeps refusing it, and ends when that tag is read again. With several refused, it names the latest one. Before engine 6.18.8 the end of the same read cleared it at once, so it showed only for a moment. "then `ERROR`" follows the thresholds in Settings every module shares: a lost connection becomes an error after `errorAfterSeconds`. All texts are in English; the full list is in Module messages. `{server}` and `{plc}` are the **Host**, `{mapping}` is the variable's code, and `{address}` and `{node}` are the tag as its row gives it. `{code}` is the driver's answer for one tag, such as `NOT_FOUND`, `ACCESS_DENIED` or `INVALID_ADDRESS`.

| When | Level | Text |
|---|---|---|
| Opening: no host | `ERROR` | `No host` |
| Opening: a collector has no schedule | `ERROR` | `No schedule given` |
| Opening: a schedule does not parse | `ERROR` | `Invalid schedule '{cron}': {reason}` |
| Opening: a tag the module cannot use | `ERROR` | `{mapping}: {reason}`, the problem as the **Type / problem** column says it |
| A connection opened | `INFO` | `Connected to {server}` |
| A read failed, and the connection was dropped | `WARNING`, then `ERROR` | `Disconnected from {server}`, then `PLC {plc} unreachable` |
| The controller refused one tag of a read | `WARNING` | `{address} returned {code}` |
| The controller refused a write | `WARNING` | `{address} refused the write: {code}` |
| A write failed, and the connection was dropped | `WARNING` | `Writing {node} failed: {reason}` |

### Example

A filter's differential pressure and its backwash state every two seconds, and an operator's limit written back when it changes, on a controller reached through the backplane:

```
Host                10.20.4.12
Port                44818
Communication path  1,0

Collector   PDIT_301      Filter1_DP:REAL         */2 * * * * ?
Collector   F_301_BW      Backwash.Active:BOOL    */2 * * * * ?
Forwarder   PDIT_301_HH   Filter1_DP_Limit:REAL
```

The two collectors share a schedule, so they are one request every two seconds. Writing `PDIT_301_HH` from a dashboard or a formula sends it to the controller as a REAL.

## Next steps

- Module catalogue
- Settings every module shares
- Module messages
- OPC UA, for controllers with an OPC UA server
- Connectivity

This page describes Data Orchester engine 6.18.8.