Critical manufacturing
an ASM PT company

Connect IoT - OPC UA Tutorial

12.0.1

November 2026

DOCUMENT ACCESS

Public

DISCLAIMER

The contents of this document are under copyright of Critical Manufacturing S.A. it is released on condition that it shall not be copied in whole, in part or otherwise reproduced (whether by photographic, or any other method) and the contents therefore shall not be divulged to any person other than that of the addressee (save to other authorized offices of his organization having need to know such contents, for the purpose for which disclosure is made) without prior written consent of submitting company.

12.0

Copy Page

Connect IoT - OPC UA Tutorial

Overview

A machine that speaks OPC UA runs a server: an address space of nodes, each identified by a NodeId. Variable nodes hold values, object nodes group other nodes, and method nodes can be called. The OPC UA driver is a client of that server: it reads and writes variables, subscribes to value changes and to OPC UA events, and calls methods. Each entry of an **Automation Driver Definition** maps to one of these:

Definition entry	On the server	Workflow task
Property	A variable node, named by its NodeId	`Get Equipment Properties`, `Set Equipment Properties`, or carried by an event
Event of `eventType` `Subscription`	One subscription, with a monitored item for each property of the event	`On Equipment Event`
Event of `eventType` `Event` or `Alarm`	An event filter for one OPC UA event type	`On Equipment Event`
Command	A method, called on the object that owns it	`Execute Equipment Command`

Table 1: How the definition maps to the OPC UA server

This tutorial is for integrators who completed Connect IoT - Basic Configuration Tutorial, which connects a controller to an OPC UA server and stops there. Here you build the integration of a furnace for a **Resource**, such as `Furnace-01`. When the furnace reports a loaded batch, the workflow reads the current setpoint, writes the batch setpoint, calls the `StartBatch` method with the batch identifier, and logs the reply. At the end, the **Automation Manager** console shows the setpoint read and the

method reply for each batch, and the setpoint node on the server holds the new value.

> **Info**
>
> For every protocol parameter and extended data field of the driver, see Connect IoT - OPC UA Driver.

Before You Start

- Complete Connect IoT - Basic Configuration Tutorial. Reuse its **Automation Protocol** for the package `@criticalmanufacturing/connect-iot-driver-opcua` and its **Automation Manager**, running in console mode.
- Have an OPC UA server that exposes the nodes of Table 2, reachable from the machine where the **Automation Manager** runs, and an OPC UA client to browse the address space and change values. The Test Orchestrator OPCUA simulator can stand in for the furnace: it hosts an OPC UA server endpoint, adds the nodes with `AddTag`, registers the method with `AddCustomCommand`, and changes a value with `WriteTag`. To run it against your MES, see Connecting to a Live MES ↗. The Test Orchestrator is a .NET test framework, currently in alpha.

Node	NodeId	OPC UA type	Access
Batch identifier	`ns=2;s=Machine.Batch.Id`	`String`	Read
Chamber temperature	`ns=2;s=Machine.Chamber.Temperature`	`Double`	Read
Setpoint	`ns=2;s=Machine.Setpoint`	`Double`	Read and write
Machine object	`ns=2;s=Machine`	Object	-
Start batch method	`ns=2;s=Machine.StartBatch`	Method with the input arguments `BatchId` (`String`) and `Temperature` (`Double`), in this order	-

Table 2: Address space of the furnace

Copy every NodeId from the address space of your own server, namespace index included. The server assigns the namespace index (`ns=`): it can change between server versions and differ between two machines of the same model.

Sequence of Steps

Step 1: Create the Driver Definition with the NodeIds

The OPC UA package ships no driver definition, because every node belongs to one particular server. Declare each entry by hand, with the NodeId exactly as the server gives it.

1. Create an **Automation Driver Definition**, such as `Furnace OPC UA`, with the OPC UA **Automation Protocol** and the **Entity Type** `Resource`, as in Create Automation Driver Definition.
2. Add the properties. The **Protocol Data Type** is the OPC UA type of the node, and the **Type** is the value the workflow receives: a `Double` node is a `Decimal` property.

Name	NodeId (Property Device ID)	Type	Protocol Data Type	Access	Extend data
`BatchId`	`ns=2;s=Machine.Batch.Id`	`String`	`String`	Readable	`arrayTy e Scal`
`Temperature`	`ns=2;s=Machine.Chamber.Temperature`	`Decimal`	`Double`	Readable	`arrayTy e Scal`
`Setpoint`	`ns=2;s=Machine.Setpoint`	`Decimal`	`Double`	Readable and writable	`arrayTy e Scal`

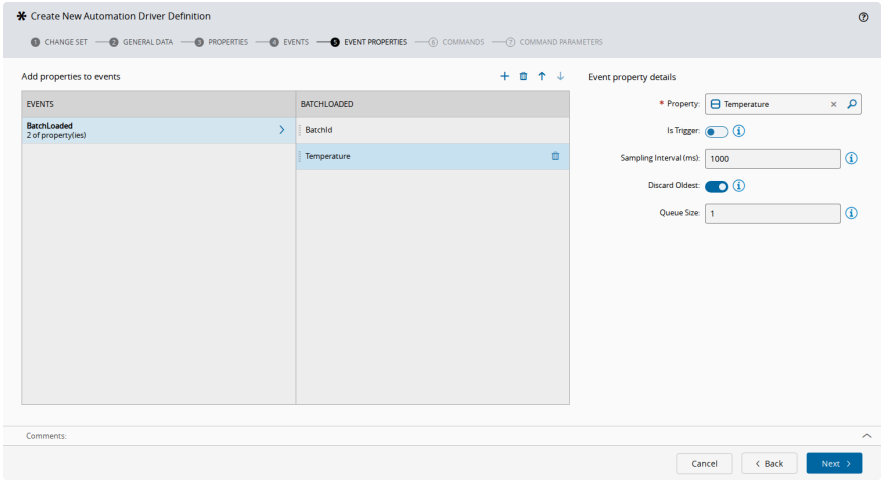
Table 3: Properties of the OPC UA driver definition

3. Add an event, such as `BatchLoaded`, with the extended data `eventType` `Subscription` and `publishingInterval` `1000`. A `Subscription` event has no node of its own, so its **Event Device ID** is not used.

4. Link the properties to the event and set their extended data:

Event property	`isTrigger`	`samplingInterval`	`queueSize`
`BatchId`	`true`	`1000`	`1`
`Temperature`	`false`	`1000`	`1`

Table 4: Properties of the BatchLoaded event



5. Add a command, such as `StartBatch`, with the **NodeId** (Command Device ID) `ns=2;s=Machine.StartBatch` and the **Object Identifier** (extended data `objectDeviceId`) `ns=2;s=Machine`.
6. Add the command parameters in the order of the input arguments of the method: `BatchId` (**Type** `String`, **Protocol Data Type** `String`), then `Temperature` (**Type** `Decimal`, **Protocol Data Type** `Double`). Select **Create**.

When the setup of the connection completes, the driver creates one subscription for the event, published every `publishingInterval`, with one monitored item for each property, sampled every `samplingInterval`. Only a change of a trigger property fires the event. `Temperature` travels with each occurrence without firing it. The first value the server reports after the subscription is the starting point and does not fire the event, and a value equal to the previous one is not a change. The driver reports the state, not every transition: changes faster than `samplingInterval`, or several changes between two publishes with `queueSize` `1`, are lost.

OPC UA calls a method on an object, so a command needs two NodeIds: the method in the **Command Device ID** and its parent object in `objectDeviceId`. The workflow

hands the arguments to the driver by name, and the driver passes them to the method by position, in the order of the command parameters. Parameters in the wrong order send each value to the wrong argument, and nothing fails when the types happen to fit.

Info

An event of `eventType` `Event` or `Alarm` subscribes to an OPC UA event type instead of value changes. Its Event Device ID is the NodeId of the event type, and `serverNodeId` names the object that emits it. `serverNodeId` has no default node: set `i=2253` for the Server object. The device ID of each of its properties is written `<namespace index>:<BrowseName>`, such as `2:Temperature`.

Step 2: Create the Controller and Set the Connection Parameters

1. Create an **Automation Controller**, such as `Furnace Controller`, with the **Scope** `Connect IoT`, the **Workflow Type** `Data Flow`, the **Entity Type** `Resource`, and one driver definition, `Furnace`, with `Furnace OPC UA`. See Create Automation Controller.
2. On the **Setup** page, open the settings of the `Equipment Configuration` task and set the parameters on the **Communication Parameters** tab:

Parameter	Value	Effect
`address`	`opc.tcp://localhost:4840`	The endpoint of the server, `opc.tcp://<host>:<port>[/<path>]`.
`user`, `password`	Empty (default)	Opens an anonymous session. Set both for user name authentication.
`securityPolicy`, `securityMode`	`None` (default)	Neither signs nor encrypts.
`certificatesRootFolder`	Default	Keeps the client certificate and the trusted and rejected server certificates. `${temp}` is the temporary folder and `${id}` the driver instance.
`defaultTransactionTimeout`	`15000` (default)	Milliseconds that one request to the server may take.

Table 5: Communication parameters of the OPC UA connection

3. Keep **Automatic Setup** enabled on the `Driver Event` task and save the controller.

The driver subscribes the events of the definition only when the setup completes. If **Automatic Setup** is disabled and no workflow sends `SetupSuccess` on `OnSetup`, the connection times out in `Setup` again and again, and every event stays silent.

When the server requires security, use the `securityPolicy` `Basic256Sha256` with the `securityMode` `Sign` or `SignAndEncrypt`. The server must then trust the client certificate: the one in `certificate` and `privateKey`, or the one the client generates under `certificatesRootFolder`. Have it trusted before the first connection. The driver reads the security settings when it receives the communication parameters, and a reconnect reuses them, so restart the controller instance after you change one.

Step 3: React to a Loaded Batch and Read the Setpoint

Create a Data Flow page, such as `Batch Start`, and add the following tasks for the driver `Furnace`:

Task	Settings to set	Inputs/Outputs to link
`On Equipment Event`	**On Equipment Event** `BatchLoaded`	-
`Get Equipment Properties`	Output **Equipment Property** `Setpoint`	`Activate` from the event `Success`
`Log Message`	**Mode** `RawText`	`message` from `Get Equipment Properties` `Setpoint`; `Activate` from `Get Equipment Properties` `Success`

Table 6: Tasks that react and read

`Get Equipment Properties` reads the node from the server when the task runs, whatever the subscription has seen, and converts the value to the **Type** of the property. The event value of `Temperature`, by contrast, is as fresh as the last sample. A read of a NodeId that does not exist does not fail: the property has no value, and its original value carries the quality `Bad`. Set the **Output Type** of the output to

`RawValue` to receive the original value, with its `quality` and `statusCode`. When a property stays empty, check its NodeId against the server first.

Step 4: Write the Batch Setpoint

Add a `Set Equipment Properties` task for the driver `Furnace`. On its **Inputs** tab, add the **Equipment Property** `Setpoint` with the **Default Value** `210`, and link its `Activate` input from the `Success` output of the `Log Message` task of Step 3.

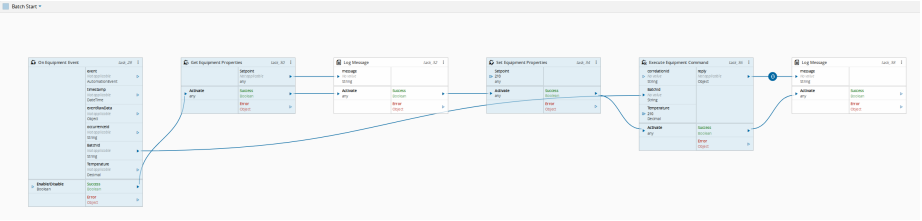
The driver converts the value to the protocol type of the node and checks its range before sending it, so a value out of range for the type fails before the server is asked. A `Bad` status from the server, for example on a read-only node, fails the task, and its `Error` output carries the status. A property that is not writable in the definition is not offered to the task unless **Allow Non Writable Properties** is enabled.

Step 5: Call the StartBatch Method

Add the following tasks to the `Batch Start` page:

Task	Settings to set	Inputs/Outputs to link
`Execute Equipment Command`	**Equipment Command** `StartBatch`; value `Temperature` `210`	`BatchId` from the event `BatchId`; `Activate` from `Set Equipment Properties` `Success`
`Log Message`	**Mode** `RawText`	`message` from `StartBatch` `reply`, through the converter `Any To String`; `Activate` from `StartBatch` `Success`

Table 7: Tasks that call the method



The `reply` output carries the single output argument of the method, an array when the method has several, or `true` when it has none. A `Bad` status fails the command, and the task emits the status on `Error`.

Step 6: Connect the Controller and Start It

Connect the controller to the **Resource** `Furnace-01` and the **Automation Manager**, with the same **Resource** for the driver `Furnace`, as in How to Create a Controller Instance. Start the **Automation Manager** in console mode.

On the Connect IoT page, the driver instance reaches `Communicating`. If it does not reach `Setup`, check the `address` and the network path to the server. If it fails in `Setup`, check that the setup phase is completed, as in Step 2.

Validate the Result

1. With the OPC UA client, write a batch identifier, such as `B-0042`, to `ns=2;s=Machine.Batch.Id`. The console shows the setpoint the server held before, then the reply of `StartBatch`. On the server, `ns=2;s=Machine.Setpoint` reads `210`, and the method received `B-0042` and `210`.
2. Change only the chamber temperature. Nothing is logged: `Temperature` is not a trigger.
3. Restart the controller instance on the Connect IoT page. Nothing is logged after the restart: the batch identifier already on the server is the starting value of the subscription. Write a new identifier, such as `B-0043`, to fire the event again.

Next Steps

To report each batch to MES, continue with Connect IoT - Talking to MES Tutorial. To react to every transition instead of the current state, have the machine count or queue the transitions in a node of its own, and mark that node as the trigger. For other protocols, see Connect IoT - MQTT Tutorial and Connect IoT - Raw File and CSV Tutorial, and for symptoms such as an event that never fires, see Troubleshooting by Symptom.

Legal Information

Disclaimer

The information contained in this document represents the current view of Critical Manufacturing on the issues discussed as of the date of publication. Because Critical Manufacturing must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Critical Manufacturing, and Critical Manufacturing cannot guarantee the accuracy of any information presented after the date of publication. This document is for informational purposes only.

Critical Manufacturing makes no warranties, express, implied or statutory, as to the information herein contained.

Confidentiality Notice

All materials and information included herein are being provided by Critical Manufacturing to its Customer solely for Customer internal use for its business purposes. Critical Manufacturing retains all rights, titles, interests in and copyrights to the materials and information herein. The materials and information contained herein constitute confidential information of Critical Manufacturing and the Customer must not disclose or transfer by any means any of these materials or information, whether total or partial, to any third party without the prior explicit consent by Critical Manufacturing.

Copyright Information

All title and copyrights in and to the Software (including but not limited to any source code, binaries, designs, specifications, models, documents, layouts, images, photographs, animations, video, audio, music, text incorporated into the Software), the accompanying printed materials, and any copies of the Software, and any trademarks or service marks of Critical Manufacturing are owned by Critical Manufacturing unless explicitly stated otherwise. All title and intellectual property rights in and to the content that may be accessed through use of the Software is the property of the respective content owner and is protected by applicable copyright or other intellectual property laws and treaties.

Trademark Information

Critical Manufacturing is a registered trademark of Critical Manufacturing.

All other trademarks are property of their respective owners.

Copyright © 2023 Critical Manufacturing. All rights reserved.