Critical manufacturing
an ASM PT company

Factory Automation

11.2

February 2026

DOCUMENT ACCESS

Public

Factory Automation

Estimated time to read: 20 minutes

Factory Automation is a factory-level workflow engine to coordinate different systems and applications to achieve high-automation.

Companies that want to deploy high automation need to have a system that listens to factory events, and coordinates and orchestrates the different systems and applications to respond to that factory event.

As factory processes become more stable and mature, manufacturers can progressively increase their level of factory automation by automating more business workflows.

Factory Automation supports the definition of workflows and links the events to workflows. It also supports hierarchical job structures as well as long running jobs which have their state and context persisted in the database.

Additionally, Factory Automation has the capability to perform error handling to recover from a variety of possible problems.

> **Note**
>
> The system accommodates fully automatic, mixed and manual scenarios.

This document will provide a quick guide for the configuration of a workflow of Factory Automation using the functionalities provided by the Connect IoT module of Critical Manufacturing.

Overview

In this example we shall define a consumer of data that is stored in a database (we can consider that it has been put there by an IoT Data Platform consumer, for example, or through the connection with a machine) as well as a controller that will handle the way that the data is received, parsed and used. Using objects provided by the Connect IoT module of Critical Manufacturing (**Automation Protocol**, **Automation Driver Definition** and **Automation Controller**), we will create the structures that will allow a user to understand and configure the environment for factory automation. For the purpose of this example, a protocol using the OPC-UA package will be configured. In order to create the appropriate structures to receive and handle the events on Critical Manufacturing, a number of steps will have to be followed:

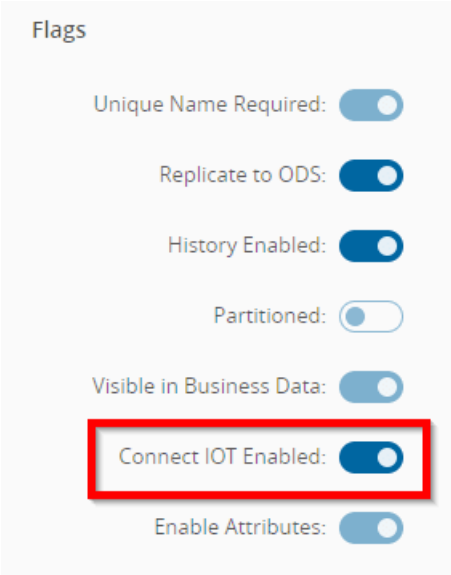
1. Create **Automation Protocols** to specify the type of protocol to use.
2. Create Connect IoT **Automation Controllers** to act as retriever of the event from the generator and another as the job creator.
3. Create **Automation Driver Definitions** that will use the **Automation Protocols**.
4. Create an **IoT Event Definition** that will enable the generation of job payloads from the received data.
5. Create a Factory Automation **Automation Controllers** that will act as job consumer.

The steps detailed above will create the objects in the system that will handle the events and the payload contained therein.

Pre-Setup

It is assumed that the Connect IoT module is licensed and has been properly installed after the main installation by following the proper procedures listed in the Installation Guide, under the Connect IoT Installation section. Two further settings are required:

- A specific configuration involves setting a property in any of the objects that will be accessed and handled by an **Automation Controller** (e.g. a **Resource**). For this, the *Connect IoT enabled* property in the **Entity Type** page for the specific entity must be active.

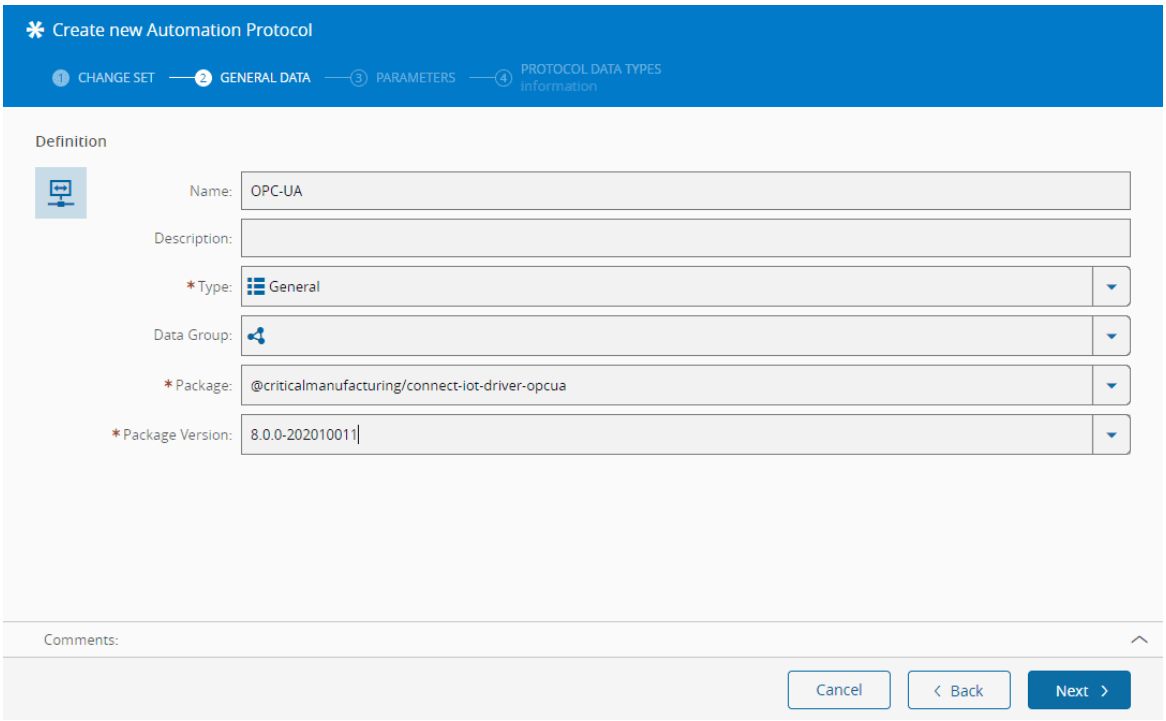


- Another setting that is required is the use of an integration user account to be used by the controller as the executor of the workflow jobs. This can be set in the User page of the **Administration** section.

With these configurations performed, let's start with the Connect IoT configuration proper:

Creating the first Automation Protocol

In the Business Data section of the main menu, navigate to the **Automation Protocol** tile and create a new entry. Provide a name for the **Automation Protocol**, bearing in mind that the name will identify the Protocol and will be used in the configuration of other entities, so make sure the name is meaningful for easy reference. Since we will be using a protocol based on the OPC-UA package, select the `@criticalmanufacturing/connect-iot-driver-opcua` package and whichever version is available and matching with the current installed version of the MES for maximum compatibility and select **Next**. An example of the values can be seen below:



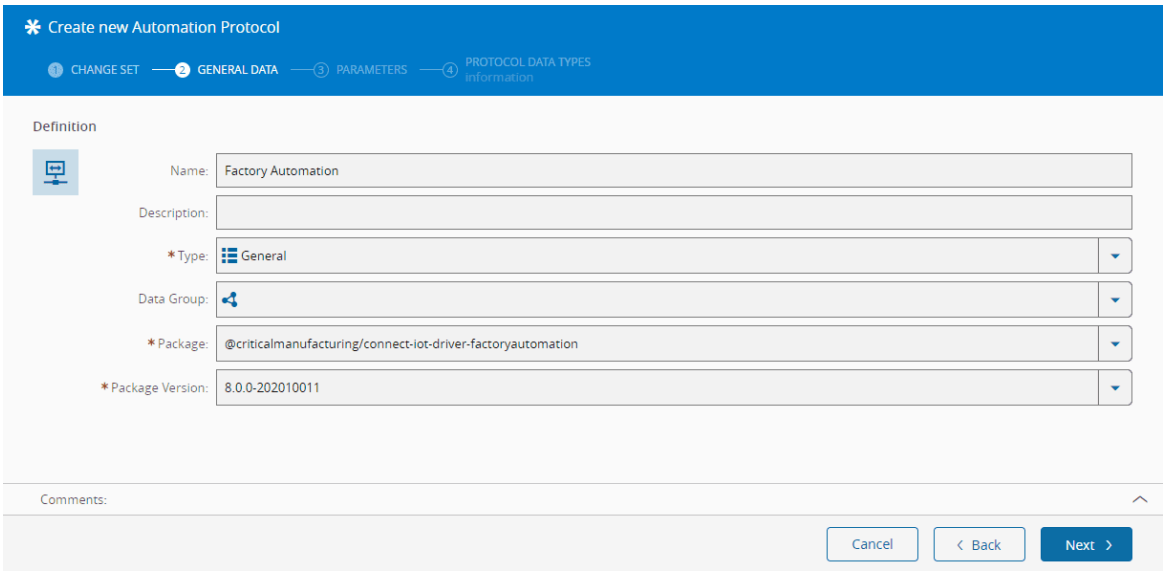
Moving on to the *Parameters* section, the values presented are the default values from the package and do not need to be altered unless specifically required. (e.g. the IT department decided that all ports must be in a specific range. Those values can be configured here and will be inherited by all the **Automation Driver Definitions** based on this **Automation Protocol**)

> **Info**
>
> More information on the OPC UA page of the User Guide.

In the last section, a list of Data Types and Extended Data (attributes per object) is displayed so that the user is aware of what is available for configuration. Pressing **Create** will finish the creation of the **Automation Protocol**.

Creating the second Automation Protocol

This second **Automation Protocol** will act upon the information retrieved by the first protocol, effectively decoupling both actions. Once again provide a name for the **Automation Protocol**, making sure the name is meaningful for later reference. This time we will use the Factory Automation package, so select the `@criticalmanufacturing/connect-iot-driver-factoryautomation` package along with the appropriate version and select **Next**. An example of the values can be seen below:



The *Parameters* and *Protocol Data Types* sections will be slightly different from the OPC-UA configuration since the drivers supply different values in both cases. Just like before, select **Create** to finish the creation of this second **Automation Protocol**.

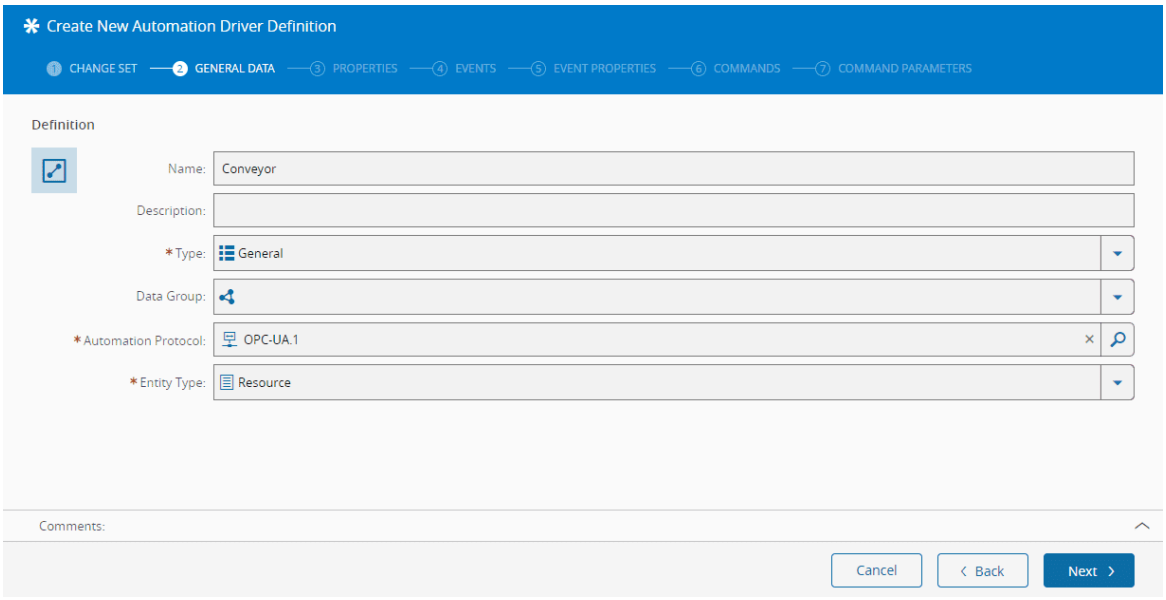
> **Info**
>
> More information on the Factory Automation page of the User Guide.

At this point, the **Automation Protocols** have been made available to the user. The next step involves creating a **Automation Driver Definition** in order to define how the machine will present the data that we want to use for each situation.

In order to simulate the generation of data coming from an OPC-UA server, we will be using the OPC-UA driver as a client and connect to a OPC-UA server running on a development machine.

Creating the Automation Driver Definition

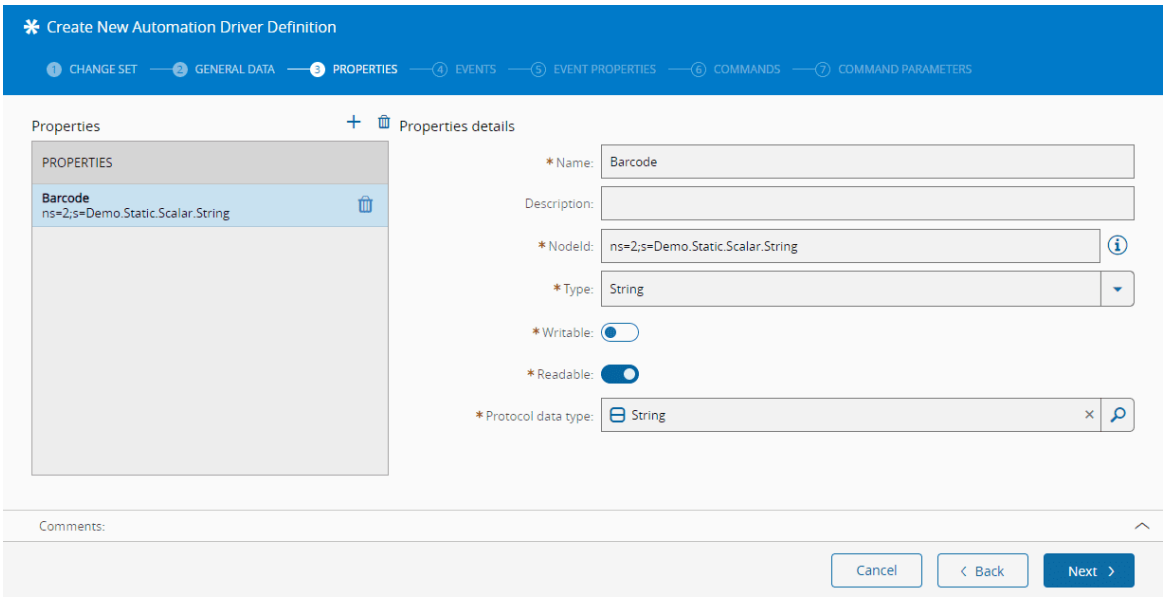
For this particular example, let's imagine a conveyor belt that is connected to the MES system via OPC-UA, with a barcode reader that detects the presence of an item when it passes on a sensor located at the end of that conveyor and writes the scanned value to a property field through OPC-UA. First of all, we should create an object that represents that same connection, in the form of an **Automation Driver Definition**. Go to the Business Data menu, select the **Automation Driver Definition** tile and create a new entry. Use a meaningful name and use the OPC-UA protocol created above as the value for *Automation Protocol*.



In the *Properties* panel, create the property that will contain the value that should be received from the barcode reader. Use the following values for the fields:

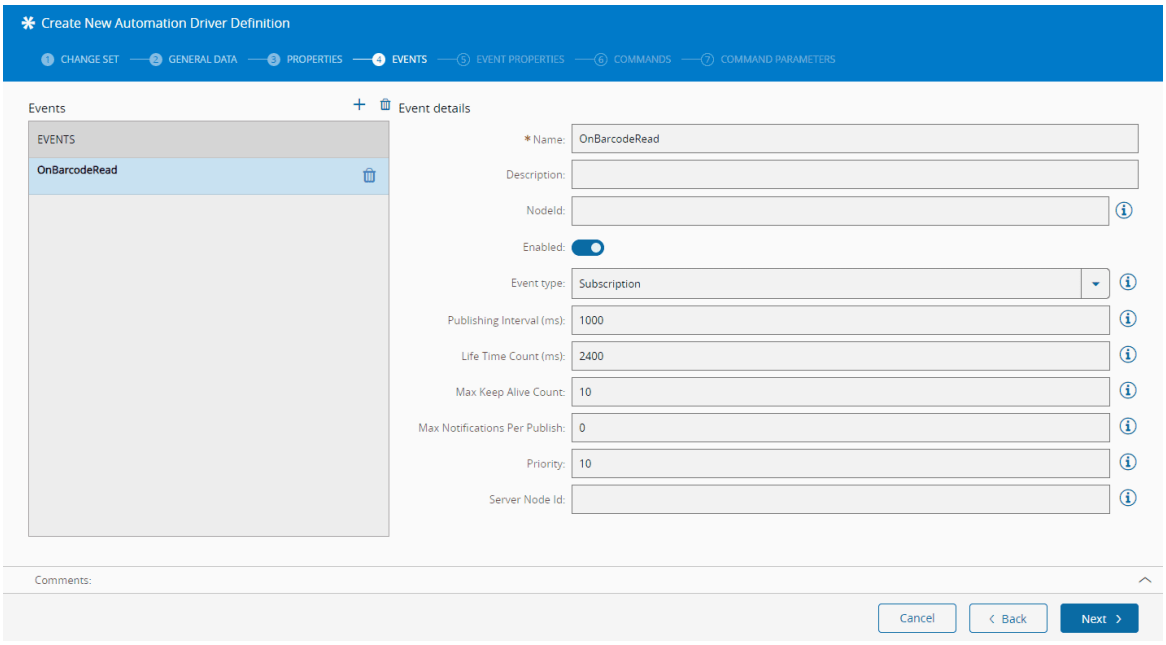
- **Name** - `Barcode`
- **NodeId** - `ns=2;s=Demo.Static.Scalar.String` (the value that can be retrieved from the OPC-UA server and is used to reference the node being written to by the barcode reader)
- **Type** - `String` (should match the value type being written)
- **Protocol Data Type** - `String` (should also match the value type being written)

The values for *Writable* and *Readable* should depend on each machine, so let's leave them untouched.



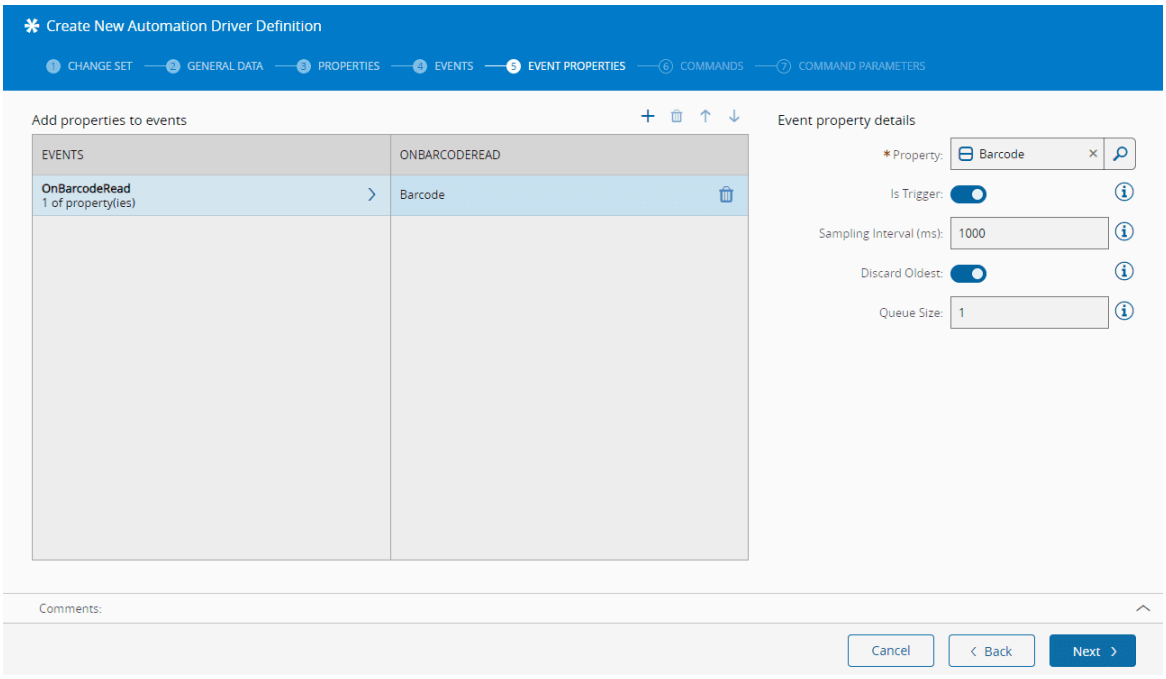
Moving forward to the *Events* panel, we will create an event that will be triggered when the value for the *Barcode* property changes. Use these values:

- **Name** - `OnBarcodeRead`
- **Event type** - Subscription (so that the MES is notified every time the value changes)
- **Enabled** - `True`



Now we shall define the actual items that we will subscribe to, in the *Event Properties* panel.

For the *OnBarcodeRead* event, add one new entry and select the *Barcode* property. Since no command will be created in the scope of this tutorial, we can jump over the *Commands* and *Command Parameters* panels and finish creating the **Automation Driver Definition**.



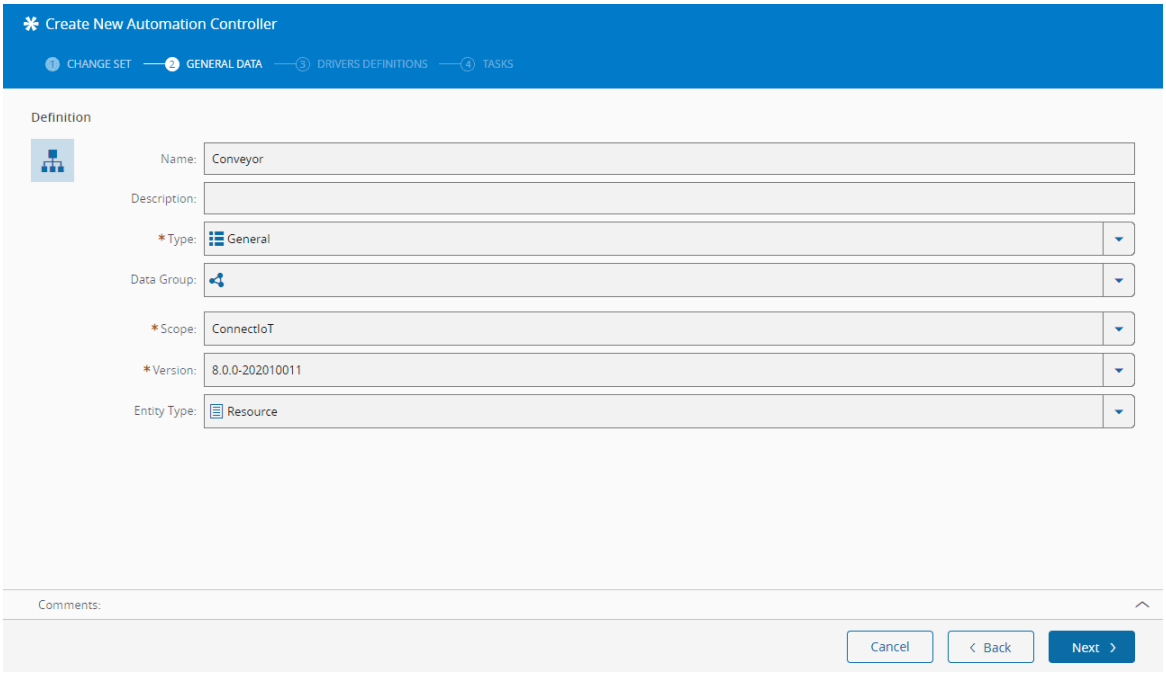
Creating the Automation Controller

We are now ready to design an **Automation Controller** that can give us the actual logic workflow that will handle the received values from the barcode reading event and act on it accordingly. Let us create one controller per each scope type (*ConnectIoT* and *Factory Automation*) Navigating to the Business Data menu again, create a new **Automation Controller** and use the following values:

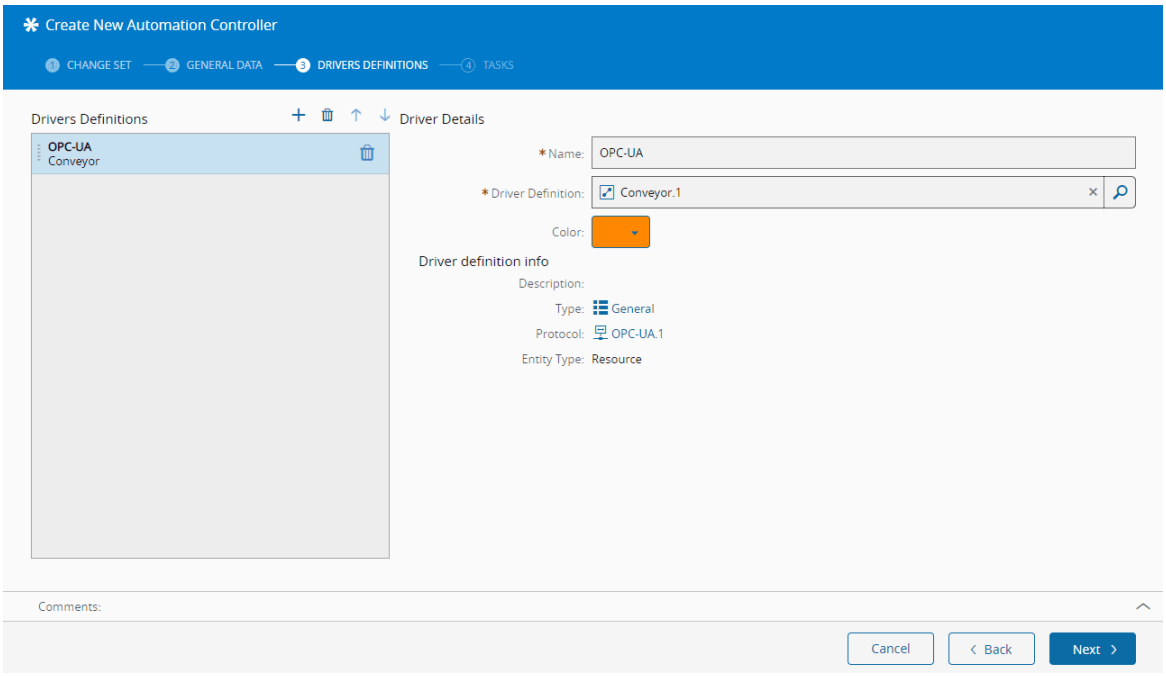
- **Name** - `Conveyor`

- **Scope** - ConnectIoT
- **Version** - The one installed with the system

> **Info**
>
> A *Connect IoT* controller will be always active and perform the logic of a workflow as soon as it is triggered, while a *Factory Automation* controller waits until it is being called through the use of Automation Jobs which are instances of those same Factory Automation workflows and get called, execute their logic and when completed are no longer used.

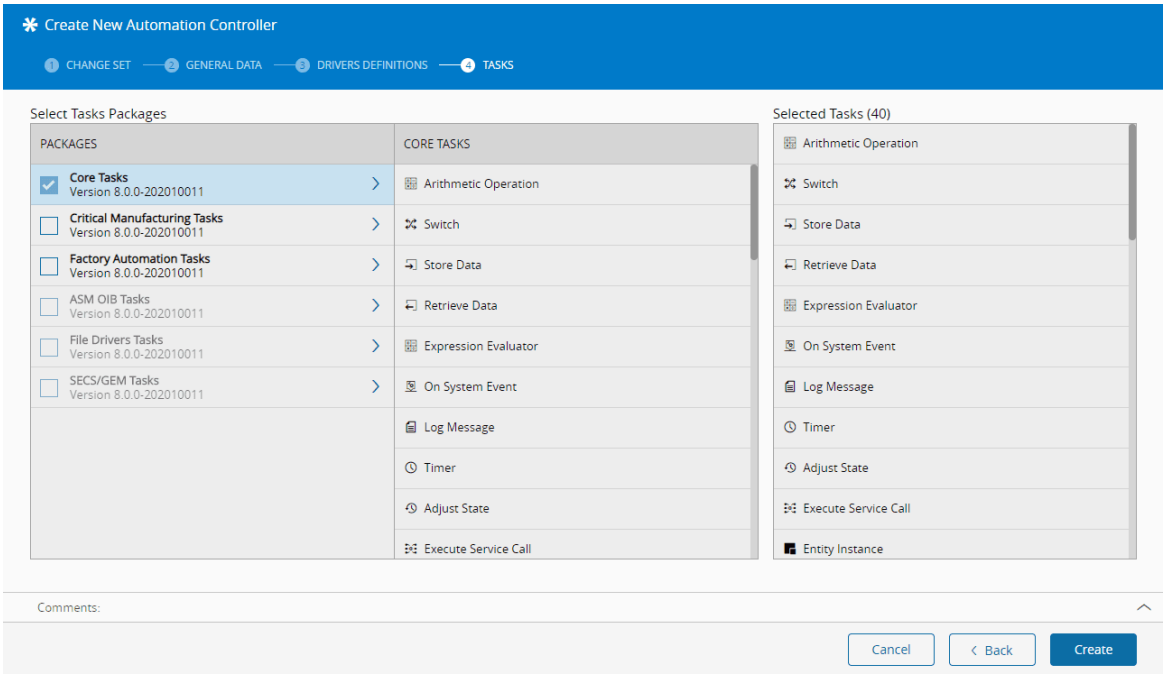


In the next panel, select the **Automation Driver Definition** created above. We can also select other driver definitions in order to enable connection to other machine types but in this case let's use the `Conveyor` driver definition created above, calling it "OPC UA Server". We will also select a different color for the tasks so that they can be easily distinguished in the workflow designer.



Moving on to the *Tasks* panel, we can select any type of task package that is specified in the ConnectIoT metadata and that are available in the system according to the installation. In this case we only need the *Core* package because all the tasks we require are located therein. We can now create the controller and start designing the workflow.

> **Warning**
>
> A user can add further packages at a later date but a task package cannot be removed from a controller once it is added, in order to avoid workflow inconsistency.



Creating the Conveyor Workflow

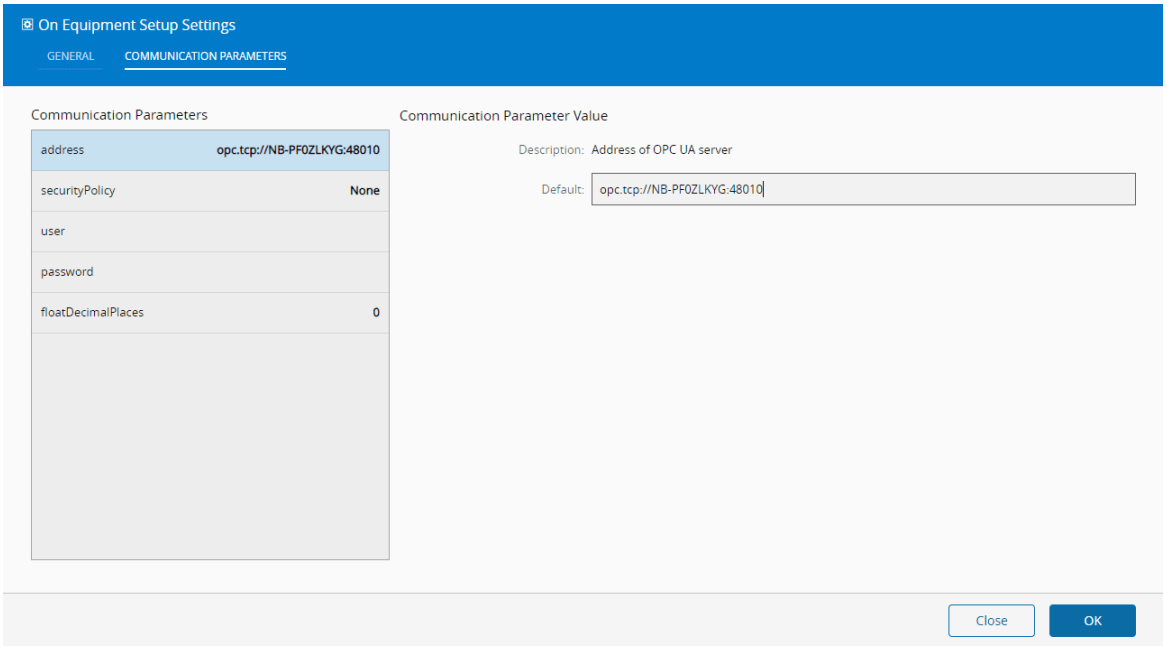
Once the Controller is created, the workflow designer is opened with the bare minimum task blocks required to be able to connect to a machine. In high-level logical terms, what is shown is the following:

- When the workflow starts setting up in the **On Equipment Setup** block and runs the startup routine (*oninitialize*), it will execute the *Connect* function to open the connection to a machine.
- When the driver enters into *onSetup* mode, it will be marked as a successful in the **Equipment Setup Result** block.

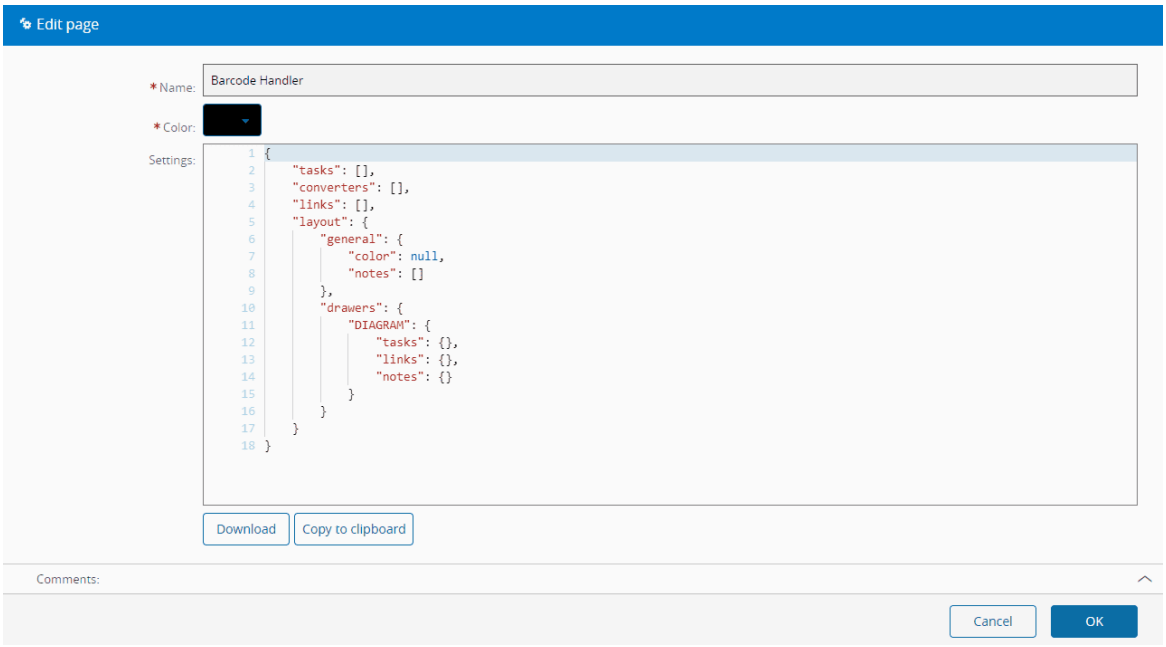
> **Info**
>
> More information on the Create Setup Workflow page of the User Guide.

Let's now tell the workflow that we want to connect to a OPC-UA server. Opening the **On Equipment Setup** block settings by pressing the three dots on top right hand corner of the block and navigating to the *Communication Parameters* tab, we will specify the address of the OPC-UA server.

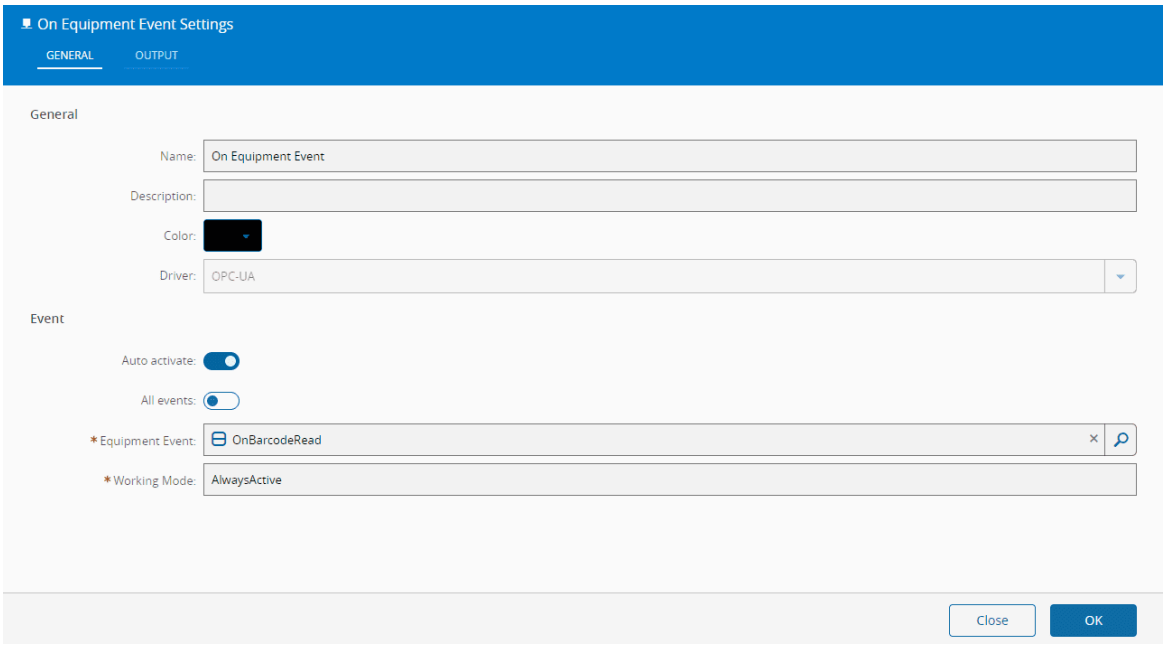
> **Info**
>
> For the purpose of this tutorial, we are using a local OPC-UA server to emulate the actual conveyor, provided by a third party application. Any compatible OPC-UA server can be used.



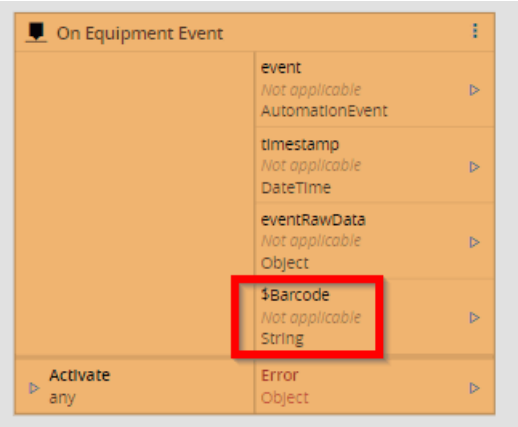
Let us then confirm that we can receive the event sent by the conveyor. First, let's create a new page in the Workflow to keep things nice and tidy. Press the + button and ✎ the new page, calling it "Barcode Handler".



In this new page, we will drag the **On Equipment Event** task tile from the right hand side *Tasks* panel to the workflow canvas. You can search for any tile name using the search box. Editing the settings of the task, we will add the Automation Event that we have configured in the **Automation Driver Definition**. Since it is the only one we have configured, it will automatically populate the *Output* tab with the *Barcode* output value we also configured before. Notice that the *Auto Activate* property is active, so whenever the value of the property is changed, the controller task will be activated.

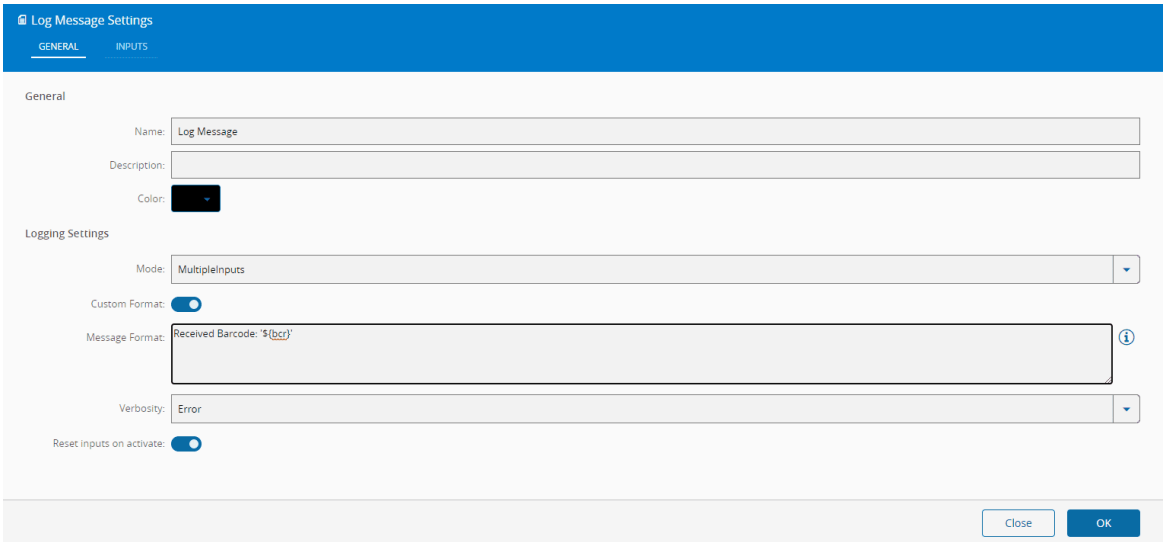


Close the wizard and notice that the *Barcode* field has been added to the task.

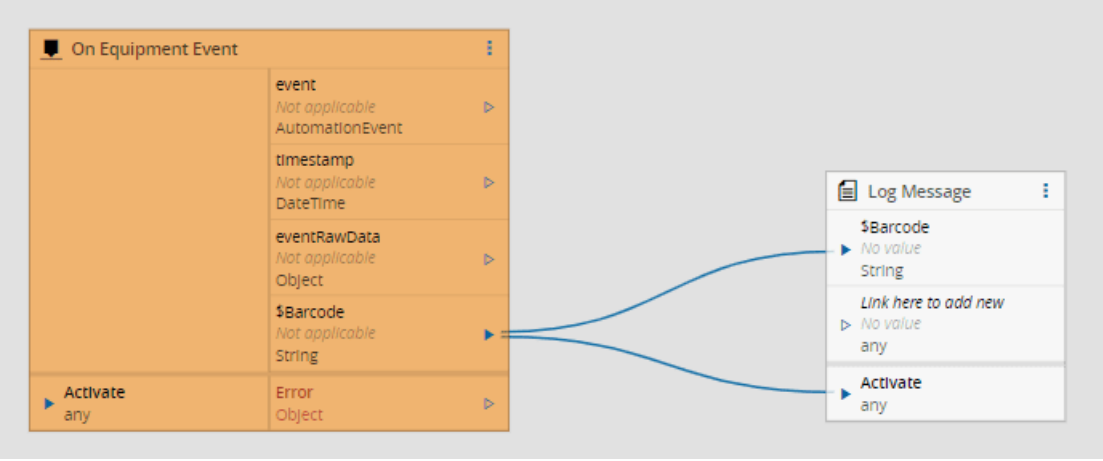


So now let's actually do something with the data that we are receiving by adding a **Log Message** task and write the value to a file. We will select that task, add it to the canvas and edit the settings:

- **Mode** - *MultipleInputs* (to enable passing the received values)
- **Custom Format** - True
- **Message Format** - `Received message ${bcr}`



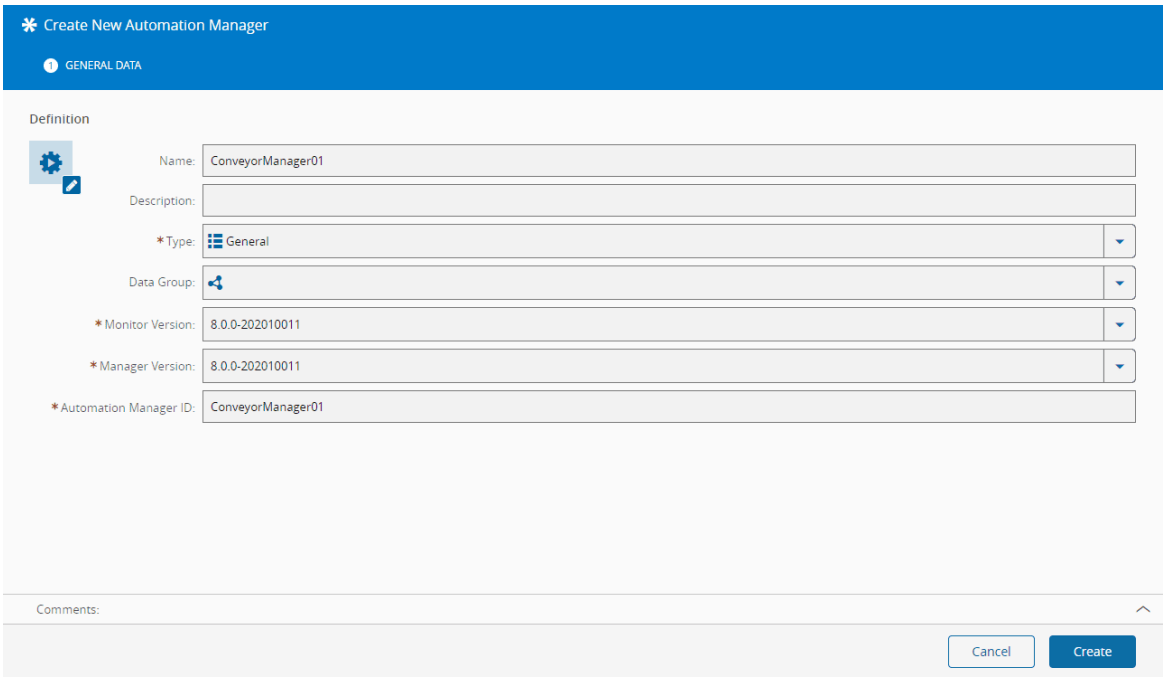
Saving the settings will allow linking both tasks through the *Barcode* property, which will automatically generate an input in the **Log Message** task.



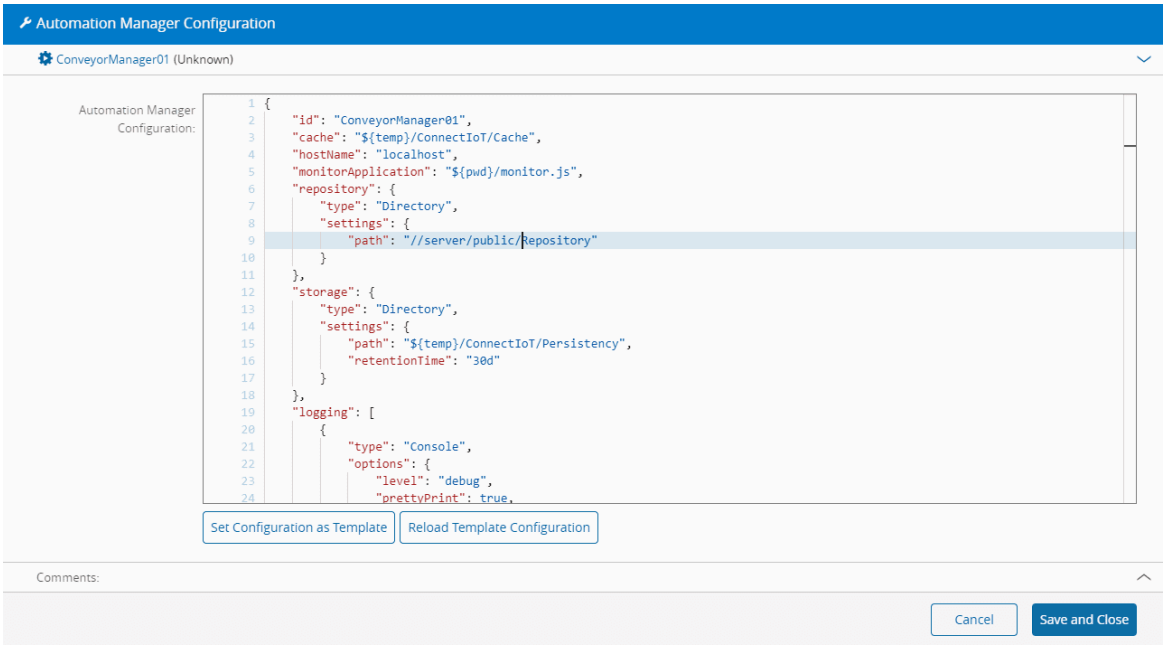
Creating the first Automation Manager

At this point we need to create an **Automation Manager** that will be a logical aggregator of the automation instances available in the system. Going once more to the Business Data menu page and creating the manager, let's use the following settings:

- **Name** - `ConveyorManager01`
- **Monitor** and **Manager** - different versions can be used although for maximum compatibility we will select matching versions
- **Automation Manager ID** - `ConveyorManager01` (unique identifier)



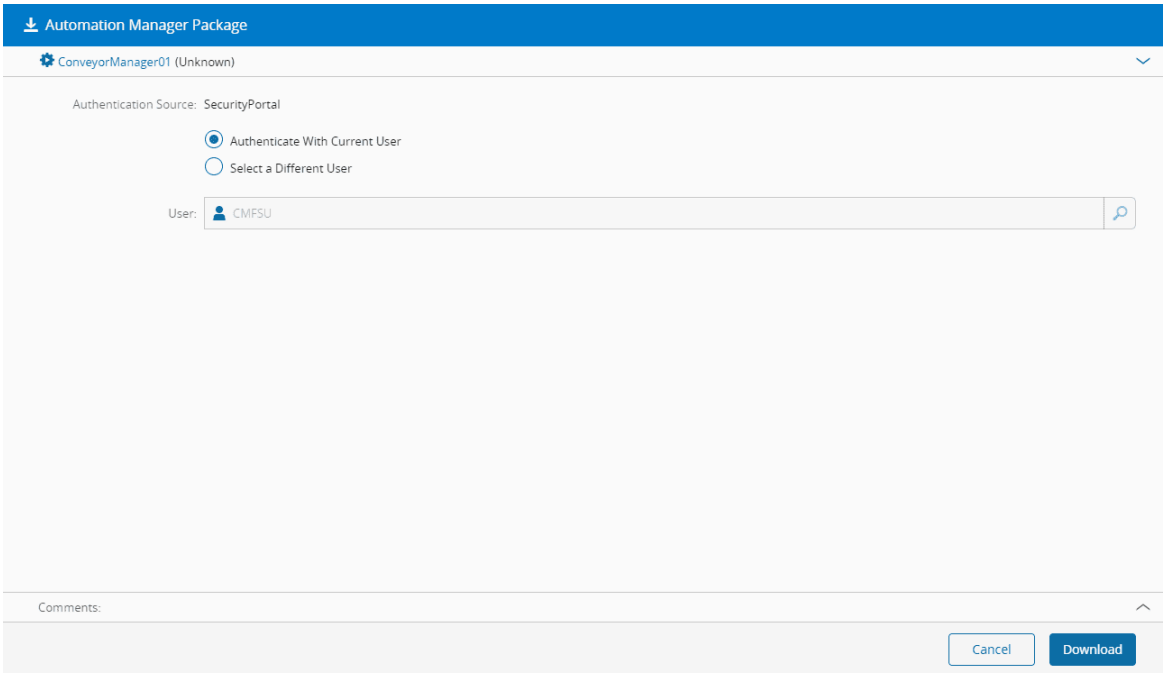
By default, the configuration is set to allow the Manager to run out of the box, populated from values retrieved from the Configuration settings of the MES as well as from the actual Connect IoT metadata.



By pressing the *Download* button, we can get a compressed file with the entire configuration cloned from the default manager but updated with our own configuration values, set when creating the `ConveyorManager01` **Automation Manager**.

Info

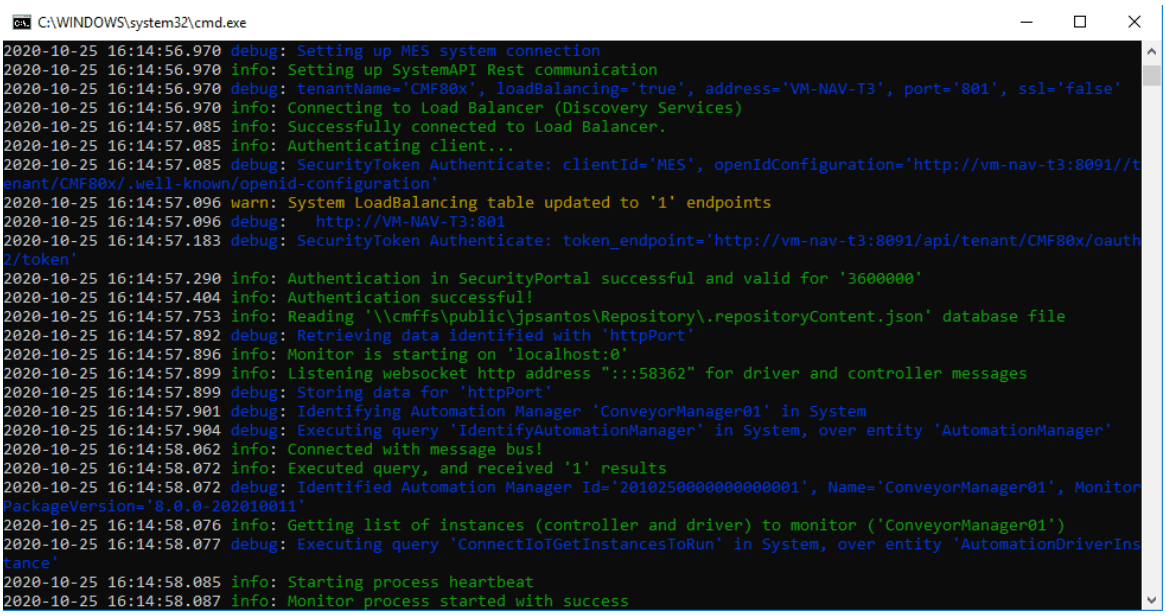
By doing this, a new token will be created in the profile of the user that extracts it and will allow it to be able to run the manager and associated workflows. For more information, see IoT Runtime Components Configuration in the Installation Guide.



The file will be downloaded and we can extract the file to a separate folder and we can run the manager by going to the `scripts` folder and run the `StartConsole.bat` file to start running the instance of Connect IoT through the **Automation Manager** we created before.

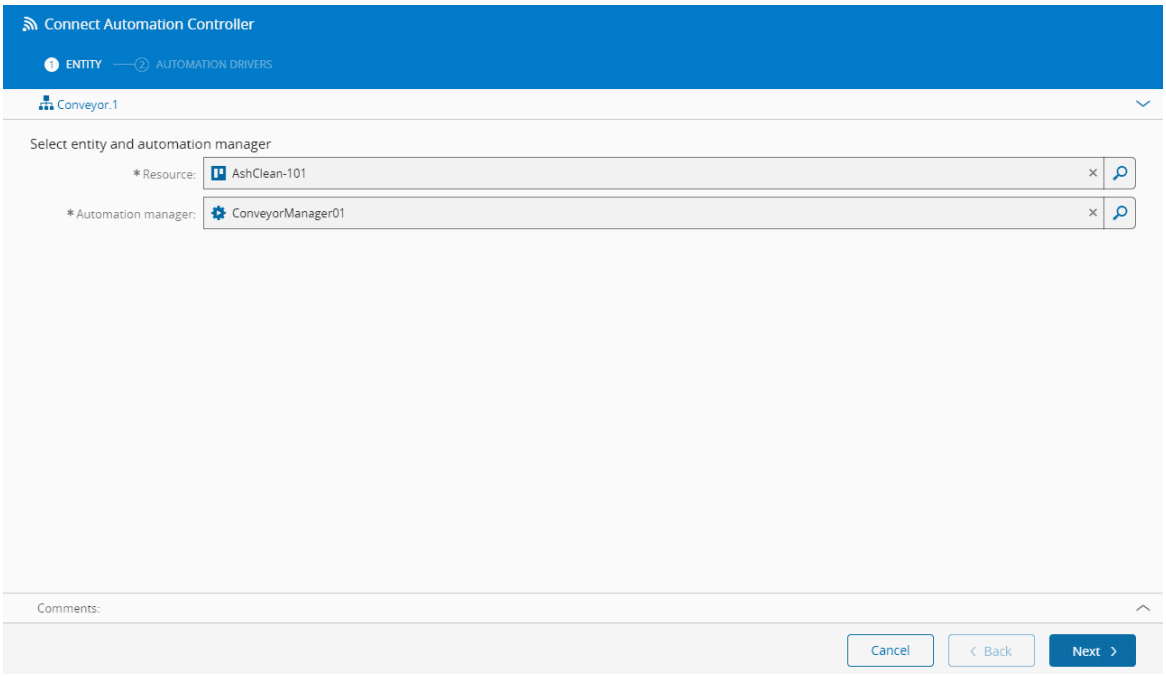
Warning

If there are no certificates available to properly authenticate the Manager, see the Troubleshooting section of the Connect IoT Installation page in the Installation Guide to solve the issue.

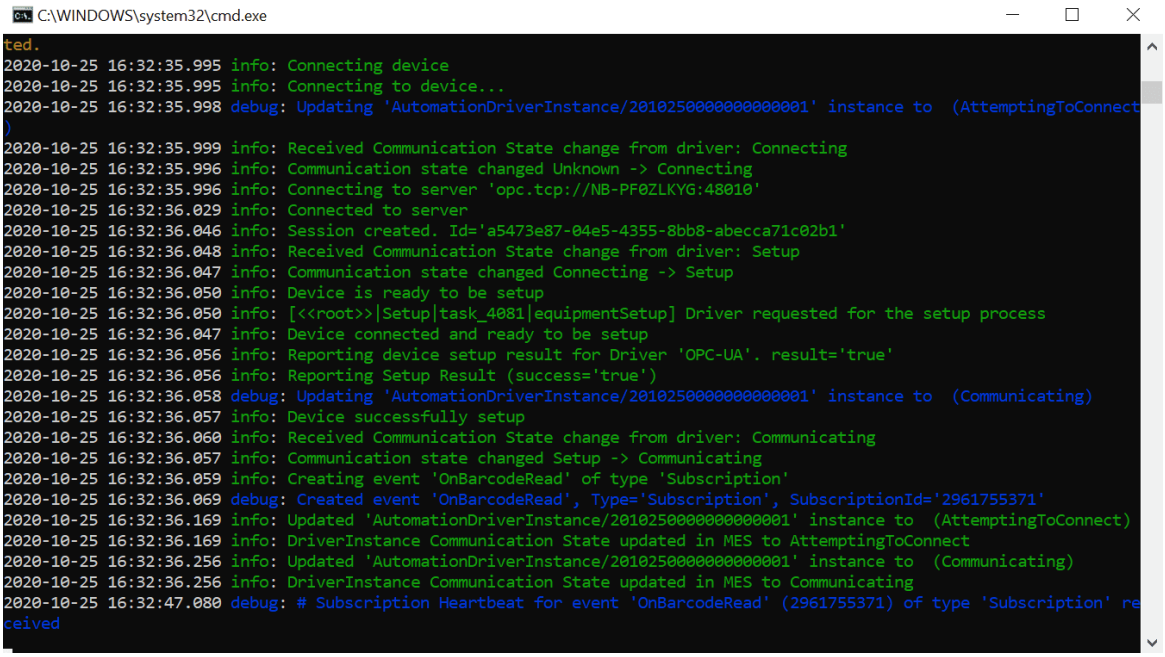


Connecting to the Manager

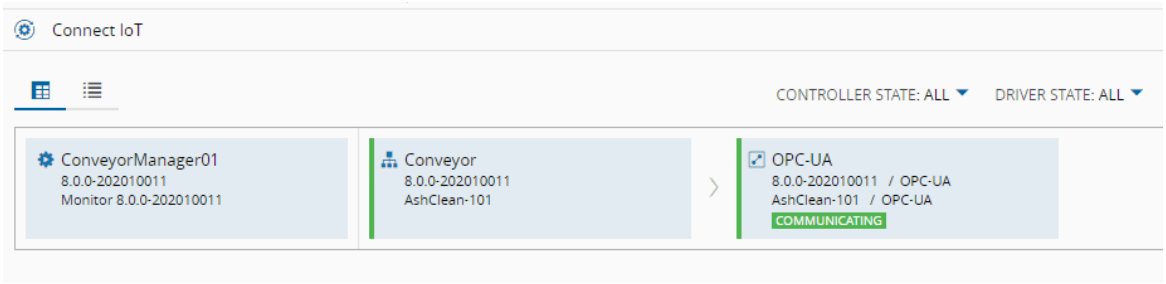
The next step involves telling the newly created **Automation Manager** what it is actually going to do. Let us then go back to the **Automation Controller** page and press the *Connect* button, specifying a specific **Resource** that we specified and using the **Automation Manager** we created above. Pressing *Next* will allow the user to select which instance will be running for each separate machine. Keeping a one-to-one relationship, let's select the same values as in the previous section and press *Connect*.



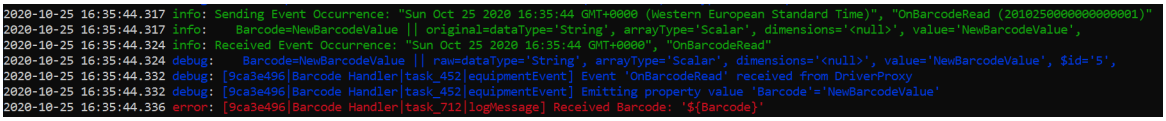
The Automation Manager will now download all the information for the entities, unpack them and will start running everything needed to run automation. Once the connection is established, we can see the status changing to `Communicating`, indicating that the connection is active.



We can also confirm that the connection is running by navigating to the ConnectIoT section of the *Automation* menu entry.



To test the connectivity, we will simulate a value being read by the Barcode Reader in out OPC-UA test client and change the property value to `NewBarcodeValue`. The value should be received by the Automation Manager since it has subscribed to any changes in that specific property and you can see that the `logMessage` entry from the **LogMessage** task being executed with the verbosity we specified, which in this case was *error*. The connection between a OPC-UA data source and the Connect IoT is then successfully established.

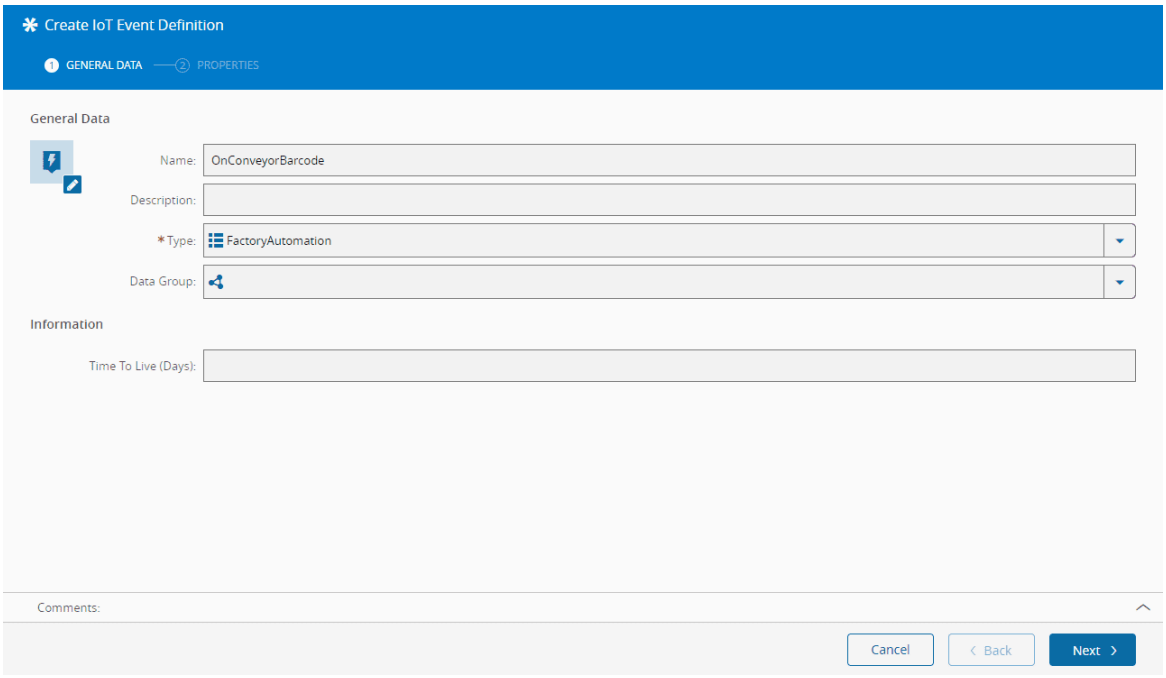


At this point we are ready to prepare the system for Factory Automation by creating the driver that will consume the data. We can now create an **IoT Event Definition** that will simulate sending the event from the machine that later on will be stored in the database for future consumption.

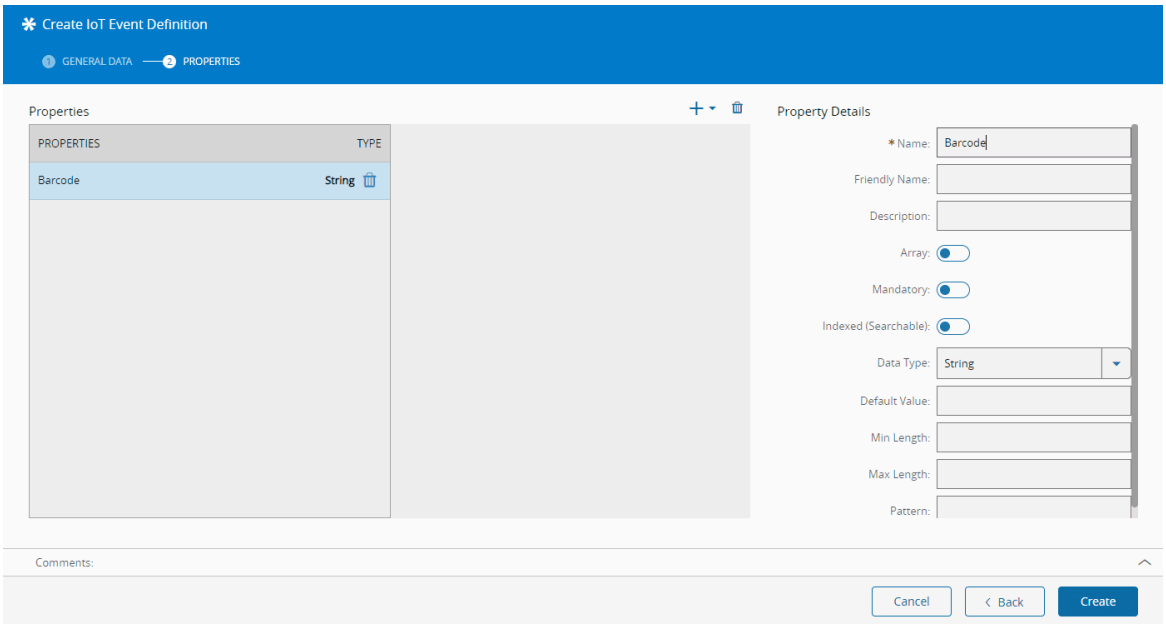
Creating the IoT Event Definition

Let's create a new entity by going to Business Data and selecting the **IoT Event Definition** tile. Let's start with the following values:

- **Name** - `OnConveyorBarcode`
- **Type** - `Factory Automation`



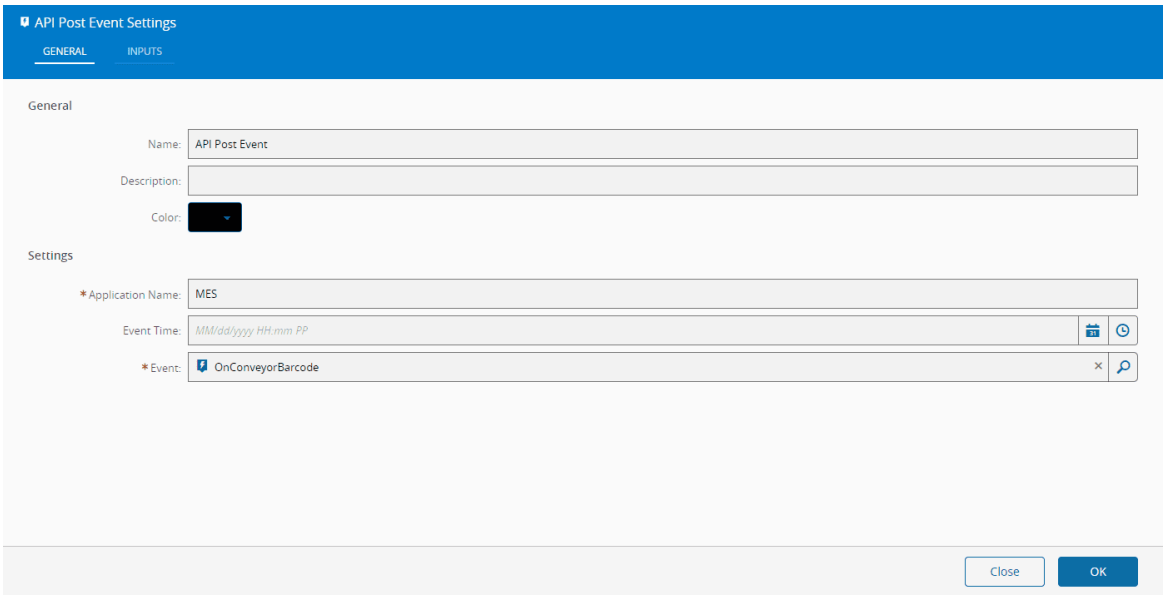
Let's also add a property called `Barcode` of type *String*.



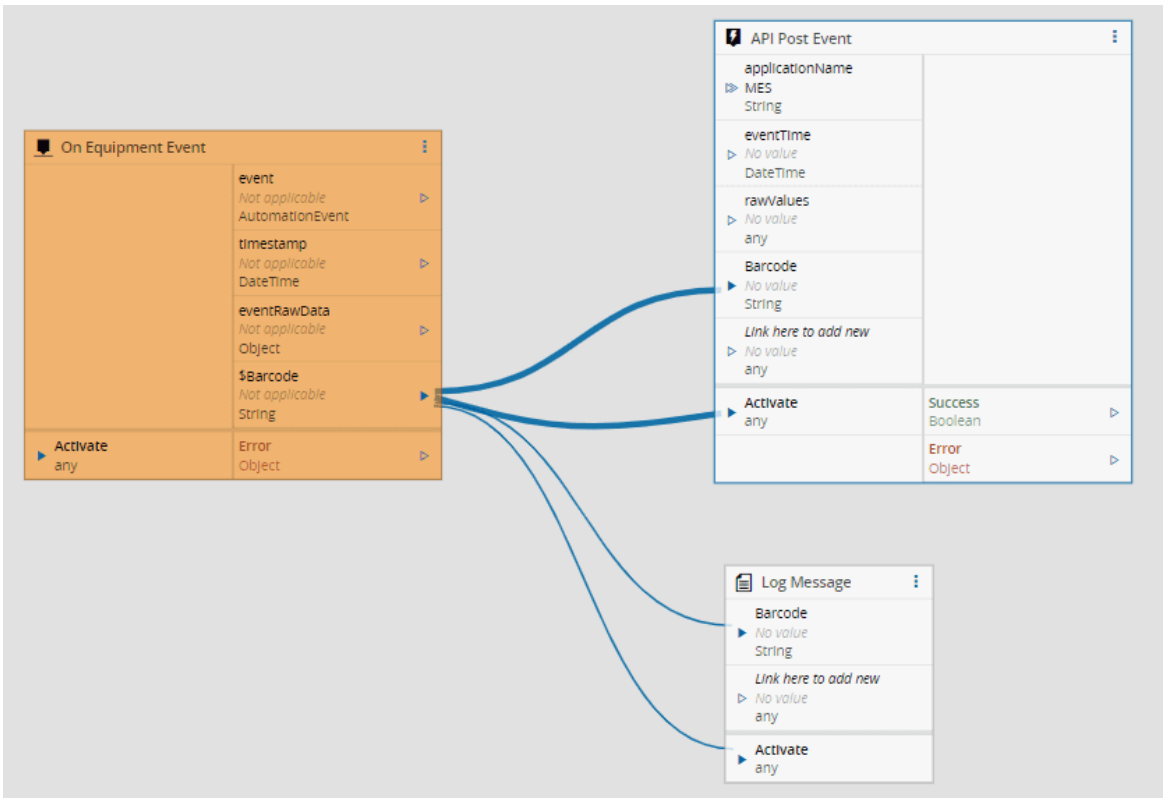
Info

For more information, see the Event Ingestion in Data Platform tutorial.

Now, to force our controller to actually send the event, let's post it every time the barcode is scanned. Going to the **Automation Controller** page, we will add a new task called **API Post Event** that will do just that. After dragging it to the canvas in the "Barcode Handler" page, we will edit the settings to select the `OnConveyorBarcode` event definition we just created as the *Event* property.



We can now link the Barcode property of **OnEquipmentEvent** that will hold the scanned Barcode and link it to the new **API Post Event** task, together with another link to activate the task.

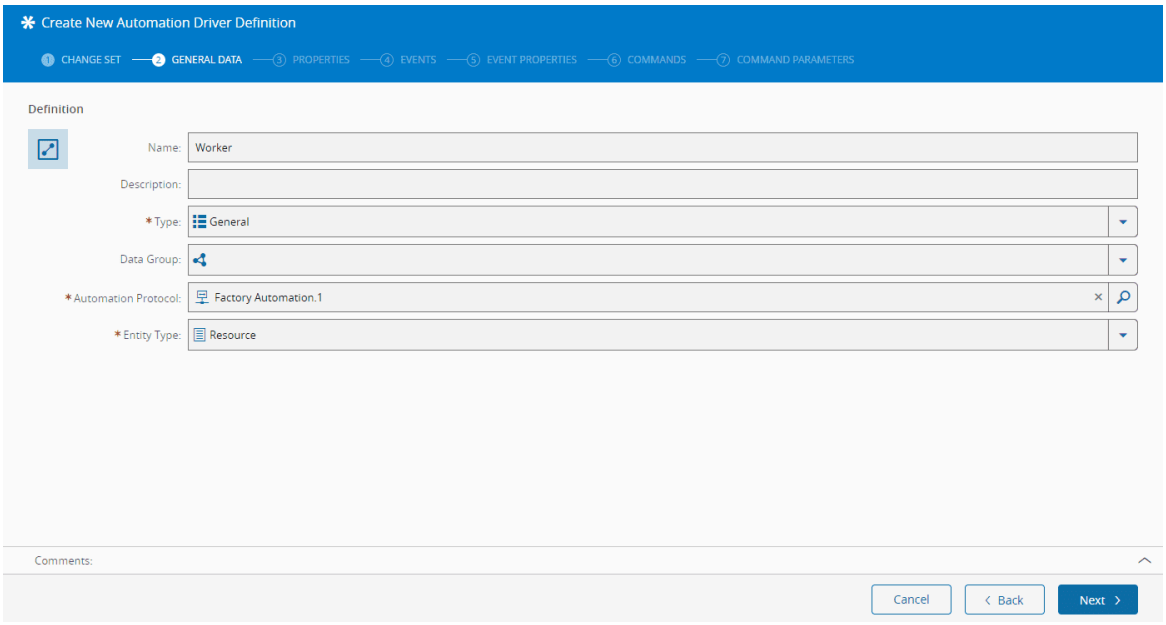


So now, going back to our OPC-UA client, if we change the value for the barcode, the event should be triggered and stored in the database.

> **Info**
>
> By default, the events will be created in a database table called `T_IoTEventQueue`.

Creating a new Automation Driver Definition

We will now create a new **Automation Driver Definition** that will form a logical bridge between the protocols. Create one using a meaningful name and using the `Factory Automation` **Automation Protocol** created above.

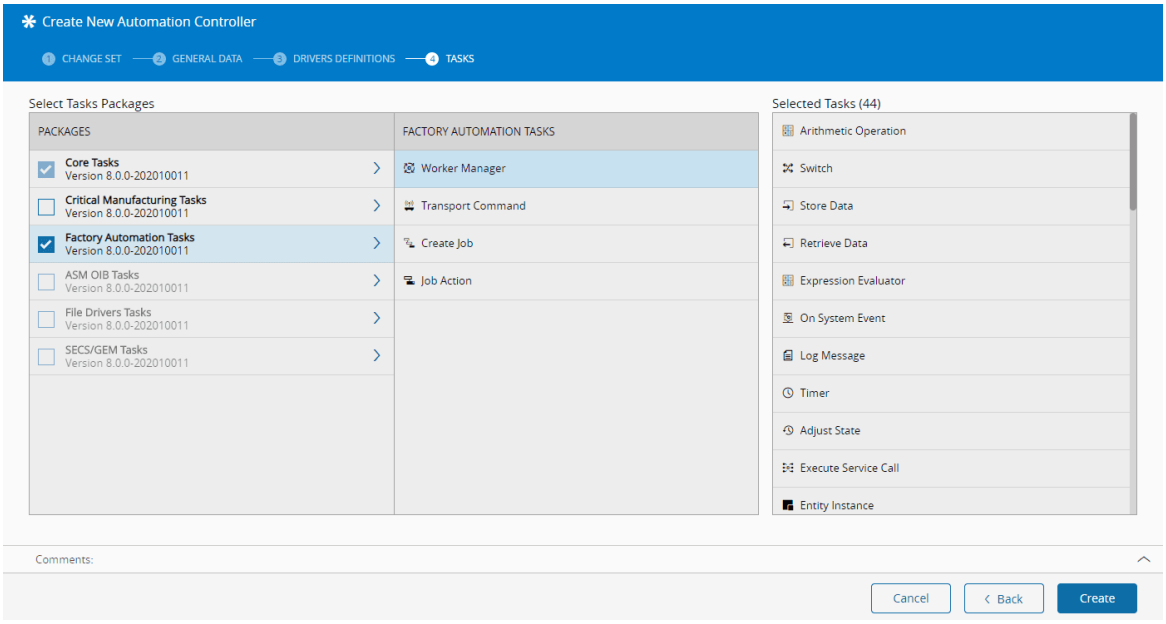


Creating the second Automation Controller

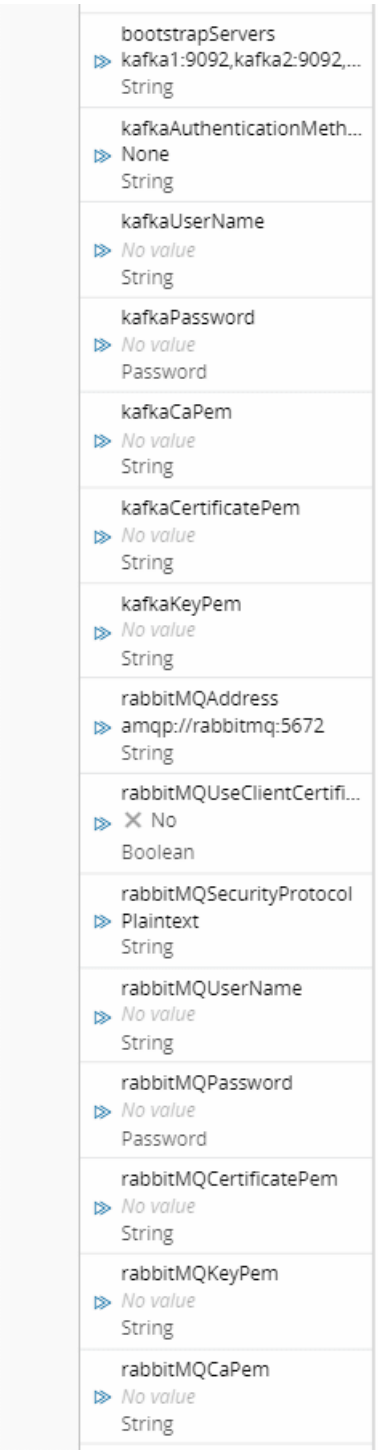
Afterwards it is time to create the **Automation Controller** to actually consume and use the data. Create a new one and use the following settings:

- **Name** - `Worker`
- **Scope** - ConnectIoT
- **Version** - The one installed with the system

In the *Drivers Definitions* page, select the `Worker` created above and name it `Database`. In the *Tasks* page, select Core and Factory Automation tasks since we will need the **Worker Manager** task which is only available in the Factory Automation package. Create the **Automation Controller**.



In this workflow, we must configure the connection to the event source (Kafka) and to the message broker (RabbitMQ) in the **On Equipment Setup** task:



- **bootstrapServers** - This is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers in a "bootstrap" Kafka cluster that a Kafka client connects to initially to bootstrap itself (kafka1:9092,kafka2:9092)
- **kafkaAuthenticationMethod** - Security protocol used (None, SASL_SSL Plain, SASL_Plain, mTLS)
- **kafkaUserName** - User name to login into Kafka
- **kafkaPassword** - Users password to login into Kafka
- **kafkaCaPem** - Kafka SSL CA Certificate

- **kafkaCertificatePem** - Kafka SSL Certificate
- **kafkaKeyPem** - Kafka SSL Private Key
- **rabbitMQAddress** - RabbitMQ address (amqp(s)://[server]:[port])
- **rabbitMQUserName** - User name to login into RabbitMQ
- **rabbitMQPassword** - Users password to login into RabbitMQ
- **rabbitMQCaPem** - RabbitMQ SSL CA Certificate
- **rabbitMQCertificatePem** - RabbitMQ SSL Certificate
- **rabbitMQKeyPem** - RabbitMQ SSL Private Key

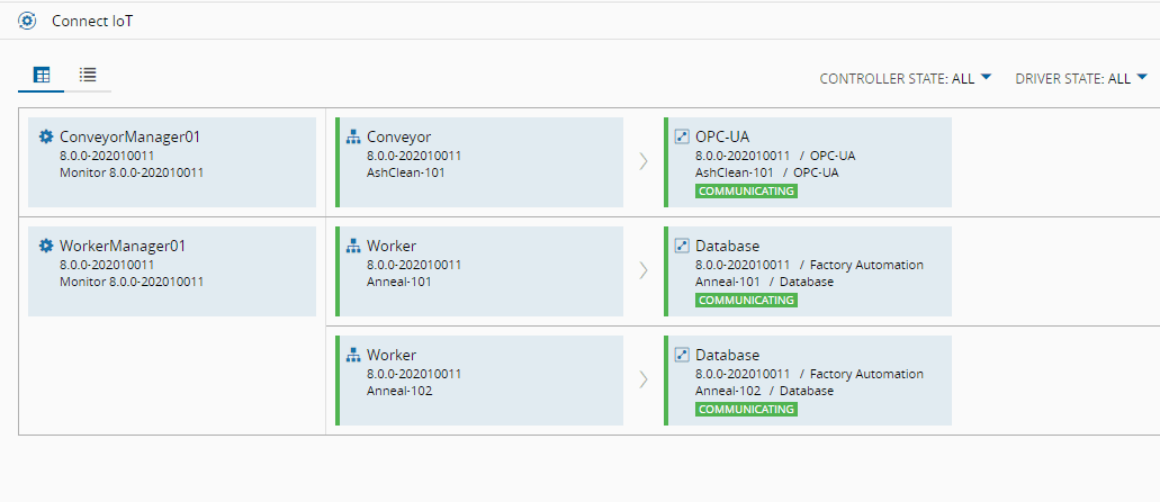
Saving these settings, we must now add a new **Worker Manager** task to the canvas. This task does not have any configurable settings but it is essential for an instance to be placed in the workflow in order to enable the proper retrieval and operation of the controller.

Warning

Worker Manager should **NOT** be removed from the workflow otherwise it will not work properly.

Creating the second Automation Manager

We can now create a new **Automation Manager** to run this new **Automation Controller**. Let's call it `WorkerManager01` and use the same value for the Automation Manager ID. Now we will download the manager and run it exactly like we did before, with the `ConveyorManager01` **Automation Manager**. By connecting via the **Automation Controller** to the running managers, we can easily scale the number of instances in operation.



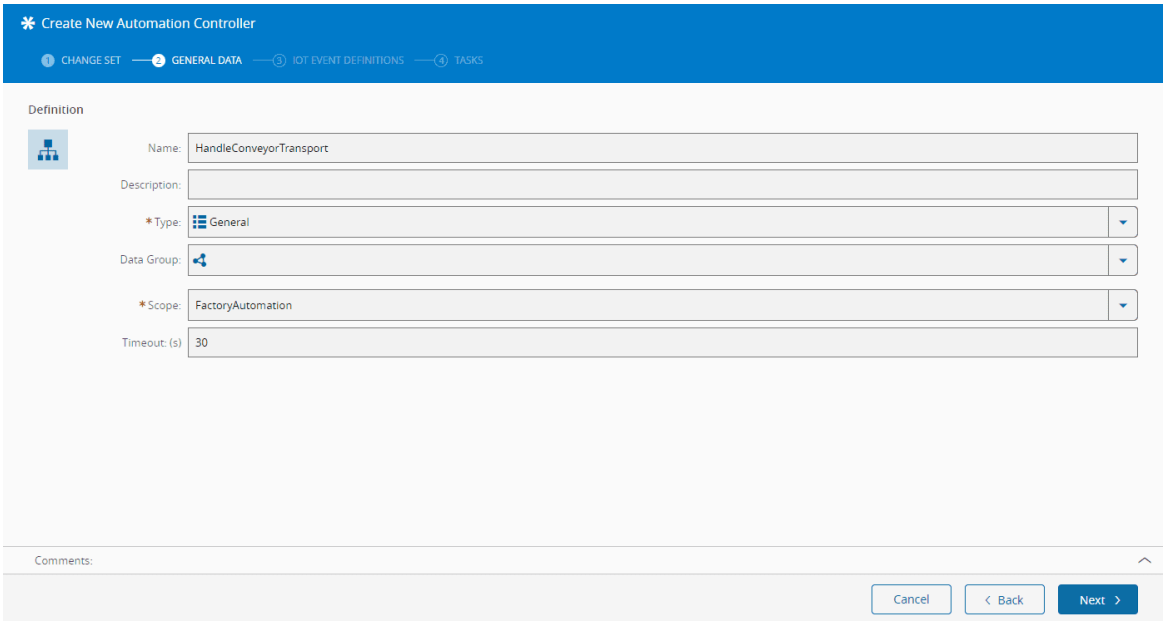
Note

The need for a second **Automation Manager** (as well as a second **Automation Controller**) is meant to allow for an easier scalability for a certain number of controllers in case we need to decouple in terms of network location as well as the need to increase the number of instances currently running when a bottleneck situation occurs and the jobs are not being adequately processed.

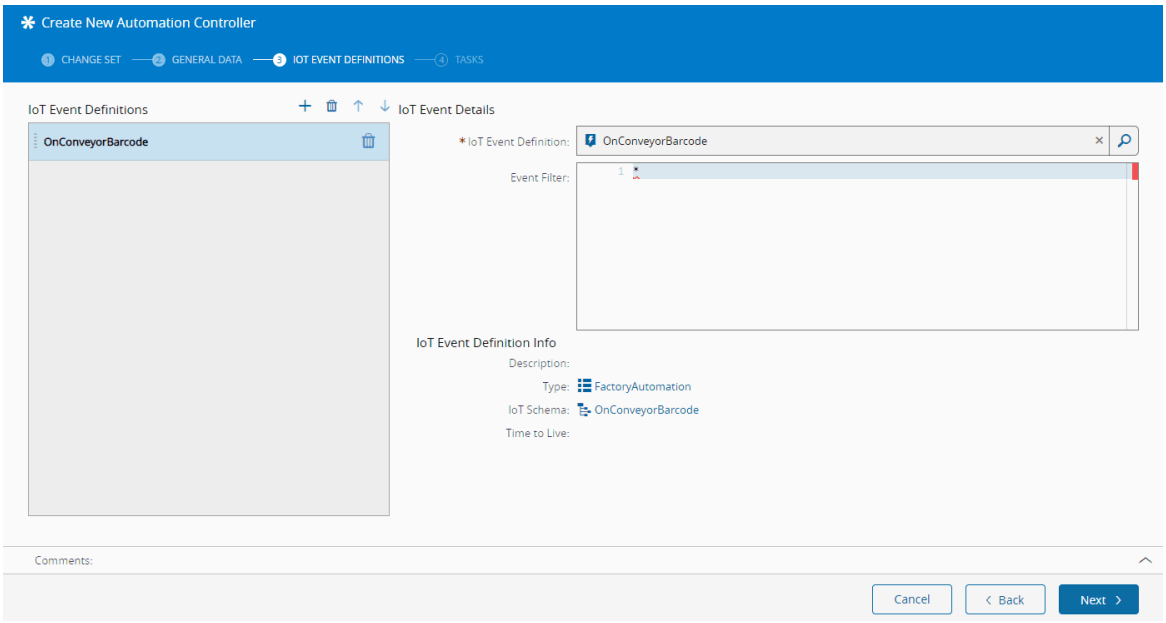
Creating the final Automation Controller

Finally, we can create the Factory Automation workflow that will process the event. Going to the **Automation Controller** section once again, we now create a new controller with the following values:

- **Name** - `HandleConveyorTransport`
- **Scope** - FactoryAutomation
- **Timeout** - 30 (seconds until the job is considered unresponsive)

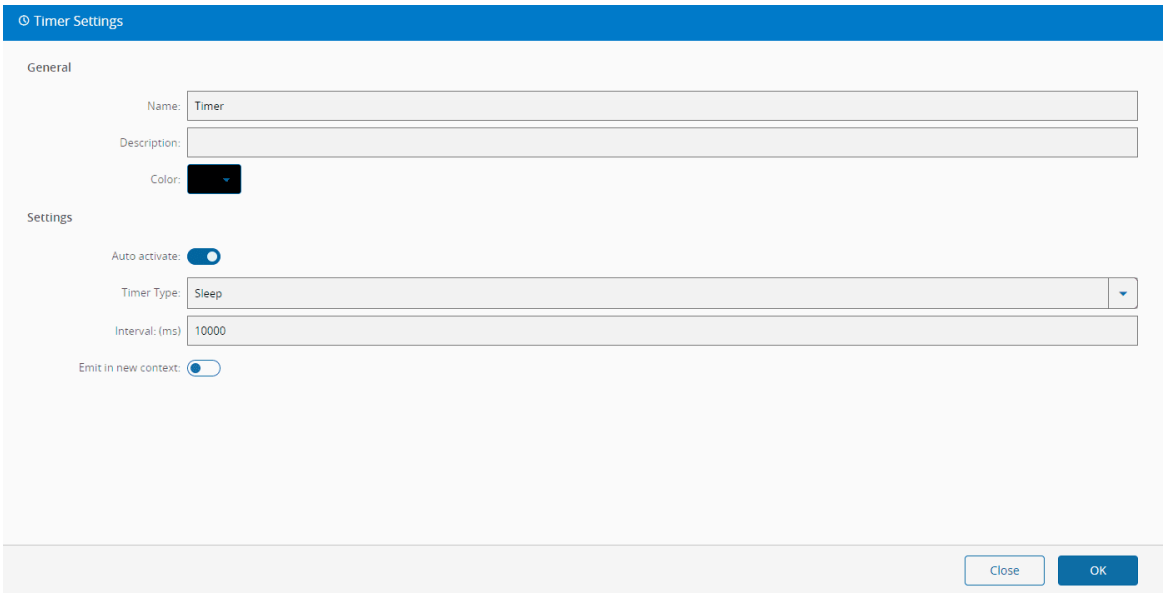


Since we changed the *Scope* to `FactoryAutomation`, instead of defining an event we will now be defining a **IoT Event Definition**. Select the `OnConveyorBarcode` we created earlier.

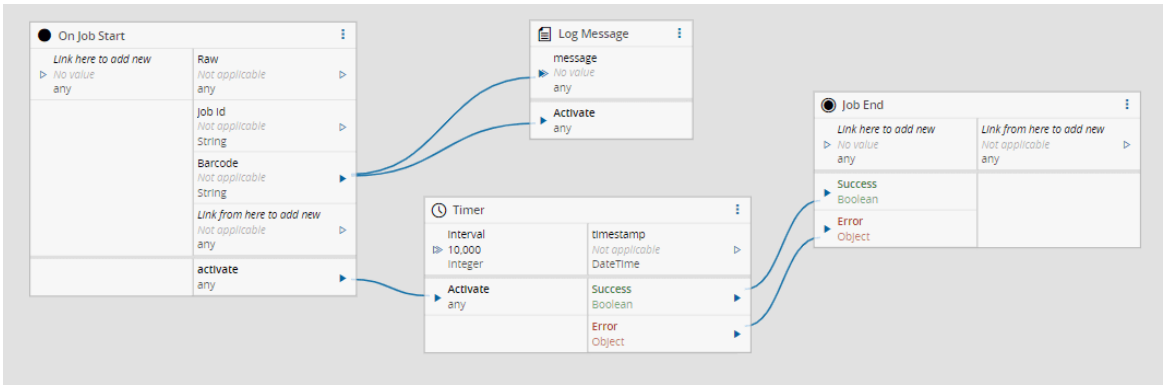


In the *Tasks* page, the Core and Factory Automation are already pre-selected due to the scope of the **Automation Controller** and cannot be unselected. Select **Create** to complete the process.

Since we already have the `OnConveyorBarcode` event definition selected, the `Barcode` property will be automatically added to the *On Job Start* task. Let us add some logging and a new **Timer** task to the canvas, configuring a sleep period of 10 seconds before doing anything. Link the *Success* and *Error* outputs of the **Timer** to the same inputs of the **Job End** task.



This last handler workflow will then look like this:

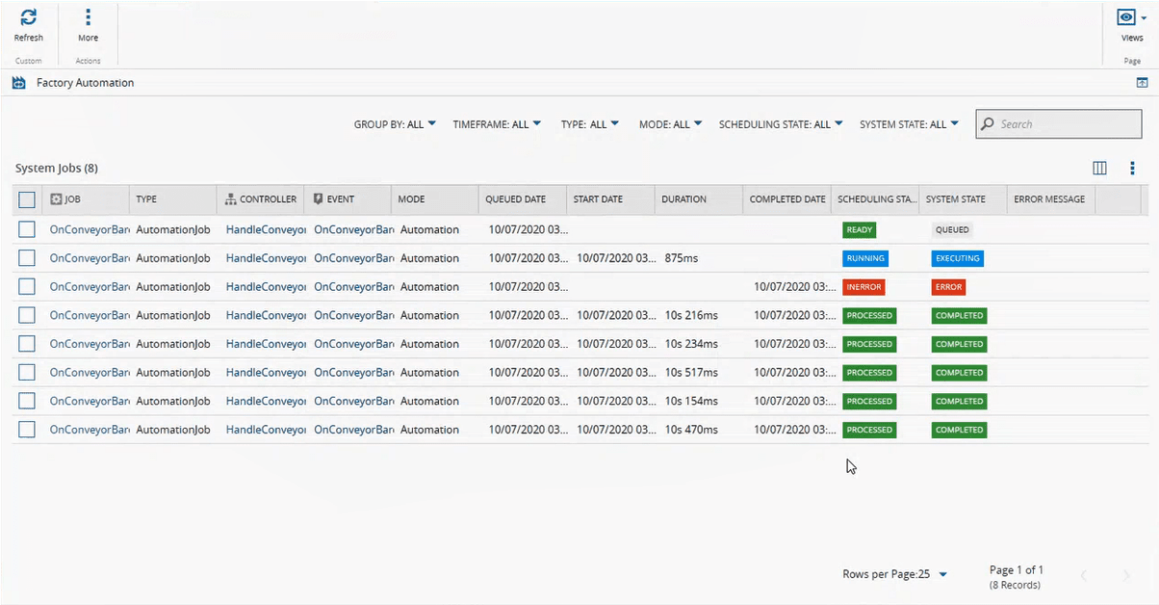


Wrapping up

So in terms of big picture, we have:

- When a barcode is read, the event is sent from the OPC-UA client and picked up by the `Conveyor` controller, which logs a message to the console. That same controller posts an event to that connects to the database and stores a job information for Factory Automation.
- The second controller picks up on those database jobs and creates a Factory Automation job.
- That job will be executed by a third controller of `FactoryAutomation` scope, that will again log the message with the value of the Barcode received and finish after the timer expires.

A list of jobs and their current status can be seen in the *Factory Automation* view, accessible from the main menu under *Automation*. If a job fails, that job can be restarted from this same page, using the buttons on the top ribbon.



Info

More information on the Factory Automation section of the User Guide.

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.