Critical manufacturing
an ASM PT company

Intermediate

10.2

March 2024

DOCUMENT ACCESS

Public

Intermediate

This tutorial builds upon the Basic configuration tutorial settings and will attempt to integrate data read from an equipment while applying some workflow logic to the retrieved value. In the basic tutorial we had an OPC-UA integration, which connected to an OPC-UA server.

The following configuration aims to support the scenario where an oven, represented by the **Resource** entity, logs a message when it reaches a given temperature, indicating the current value. Then it will grab that information and post it to a Data Collection.

> **Note**
>
> During this tutorial, the **Automation Manager** will run in console mode in order to highlight the most important events as they take place.

Automation Driver Definition

Let's go over the **Automation Driver Definition** Oven DD and set the properties and events that are needed to support the scenario.

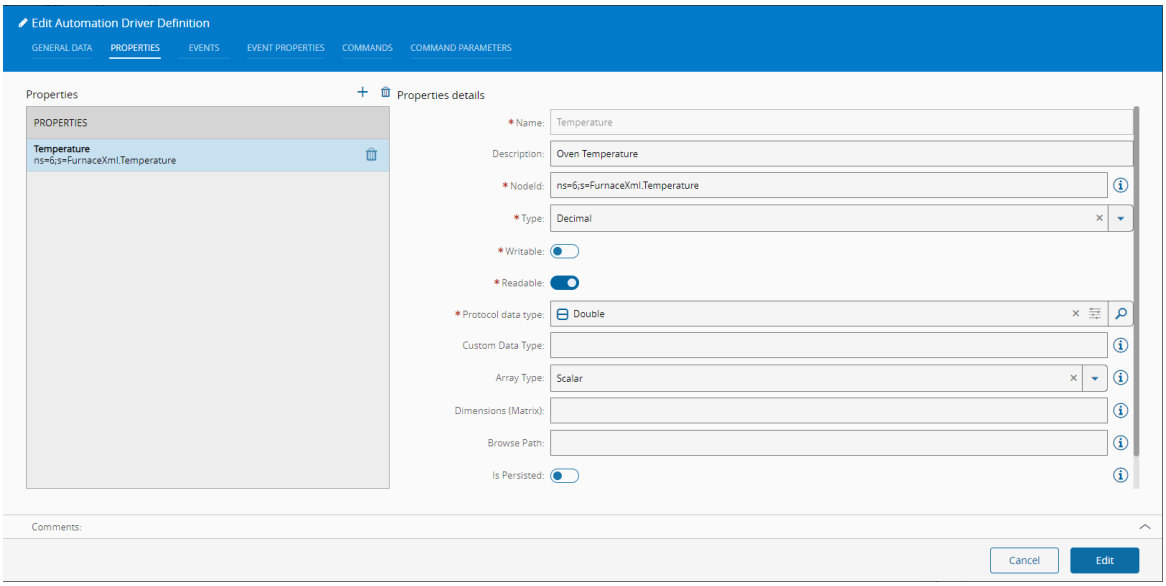
> **Note**
>
> Note that to do this in a real life context, it is important to have a general knowledge about the protocol, the equipment itself and its related documentation.

Properties

Go to `Automation Driver Definition`, select the Oven DD and then select `Edit`

To add the Temperature property:

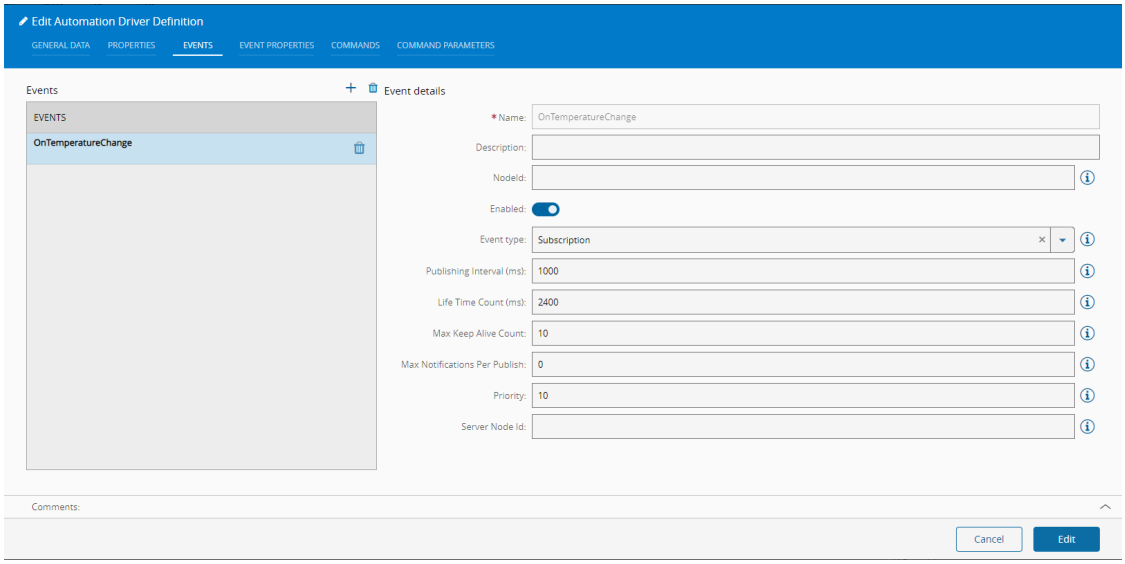
1. Skip the General Data step
2. Add a new entry to the list of Properties by selecting +
3. In the Property details, provide:
   - A name that represents the Property name
   - A description
   - The NodeID - identification of the Property - check the equipment documentation for the actual identification of the property on the equipment
   - The type (for classification and reporting purposes)
   - The `Writable` and `Readable` flags
   - The data type of the Property in OPC UA format - check the equipment documentation for the actual data type of the property on the equipment
4. Select `Events` step



Events and Event Properties

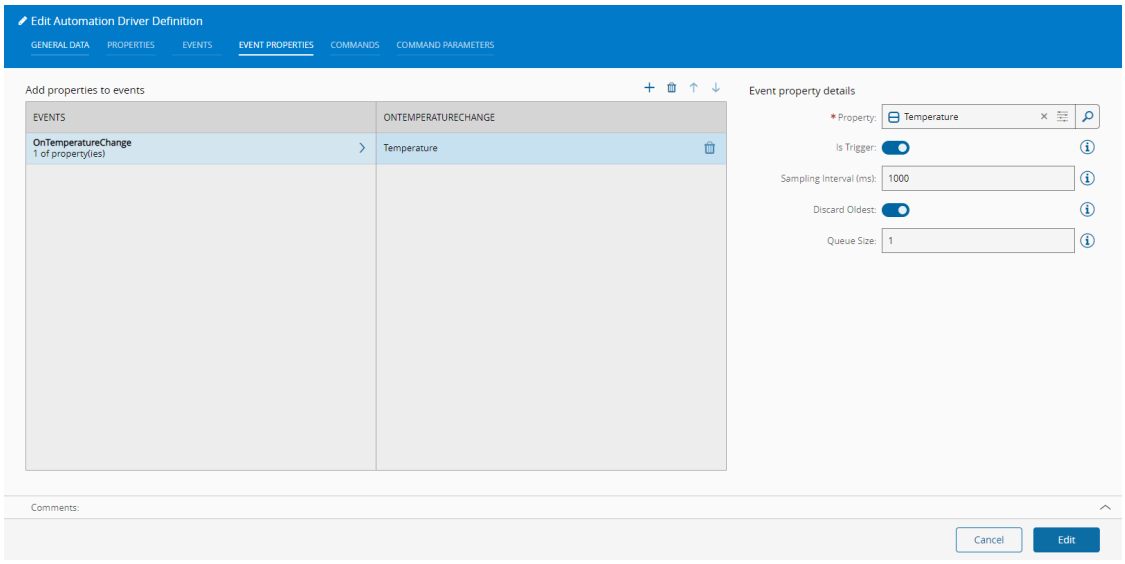
Now we need to add the Event On Temperature Change, that based on a technicality on the Protocol, we know that every time a Property value changes, an Event occurs.

1. Add a new entry to the list of Events by selecting +
2. In the Event details, provide:
    - A name that represents the Event name
    - A description
    - The `Enabled` flag
    - The Subscription Event Type
    - The Publishing Interval in milliseconds
    - The Life Time Count in milliseconds



3. Select `Event Properties` step
4. Select the previously created Event and select + to add a Property to the Event
5. In the Property details, provide:

- The previously created Property
- The `Is Triggered` flag. In this case we only have one tag that we are monitoring, but with multiple tags it's very important to accurately decide what tag will you use as your trigger.
- The Sampling Interval in milliseconds
- The discard oldest flag
- The queue size

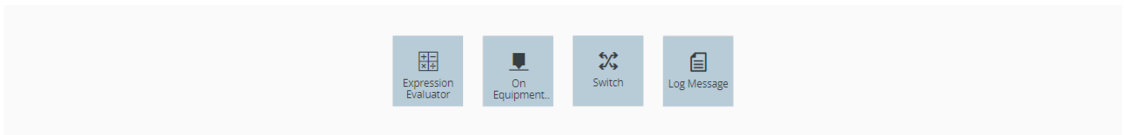


With this settings, when the `OnTemperatureChange` event is triggered, MES will receive the Temperature value, at the time it did happen. This tutorial does not focus on the equipment commands so skip the Commands panel and select `Edit` to complete your changes.

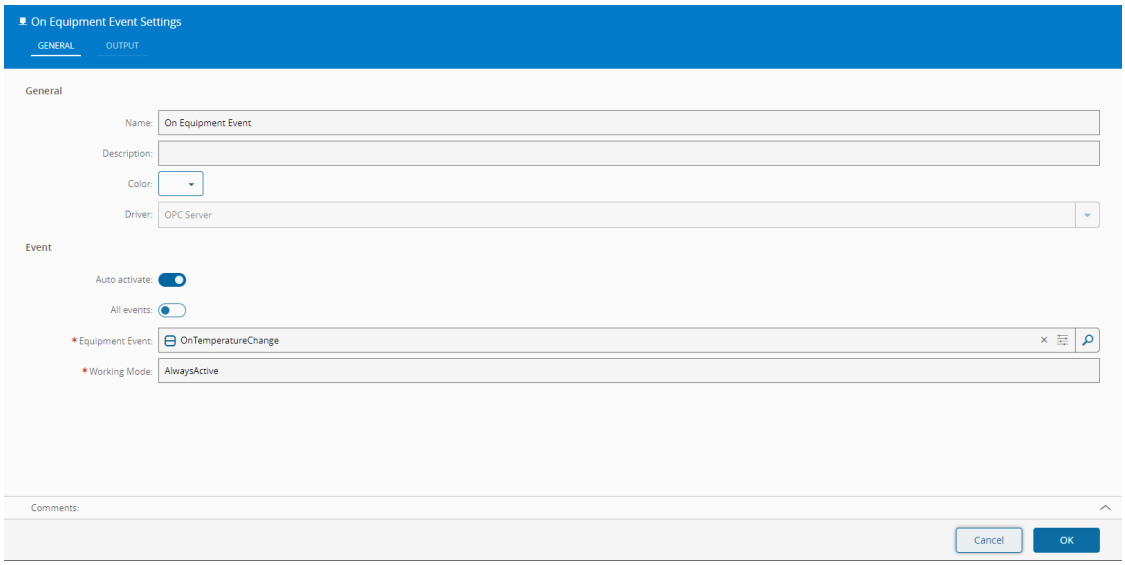
Automation Controller

Let's go over the **Automation Controller** Oven Controller and define the logic that will support the described scenario.

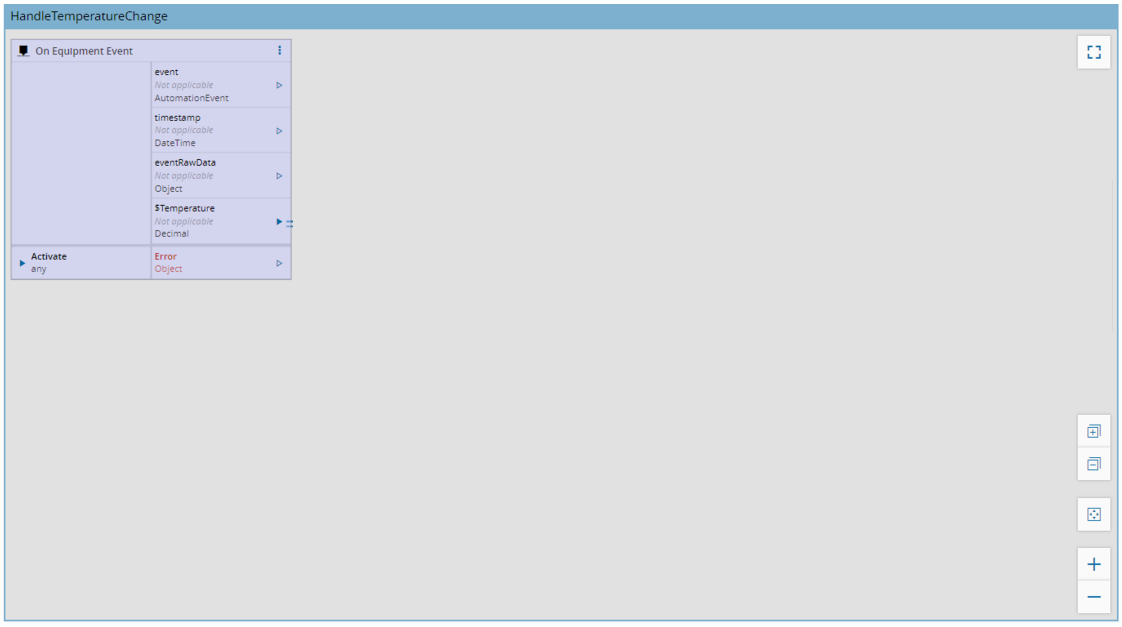
1. Go to `Views > Workflow`, and in the page right panel select + and edit, then and rename the page to a more friendly name, for example HandleTemperatureChange
2. Drag and drop the following tasks:
   - `On Equipment Event`: to listen to the Event "OnTemperaturChange", and to retrieve Temperature values
   - `Expression Evaluator`: to assess if the Temperature is above 200 degrees
   - `Log Message`: to print the message into the console, in case the Temperature evaluation result is true



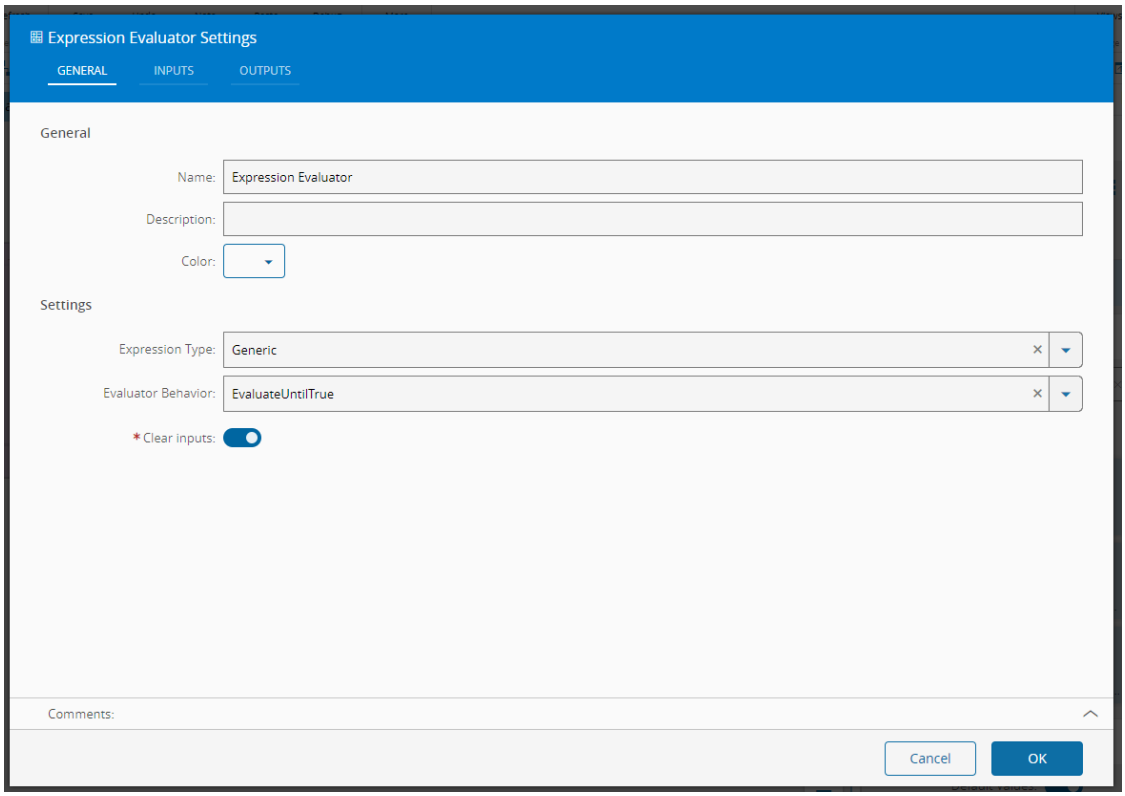
3. Go to the `On Equipment Event` settings and for the Equipment Event, select the Event OnTemperaturechange.



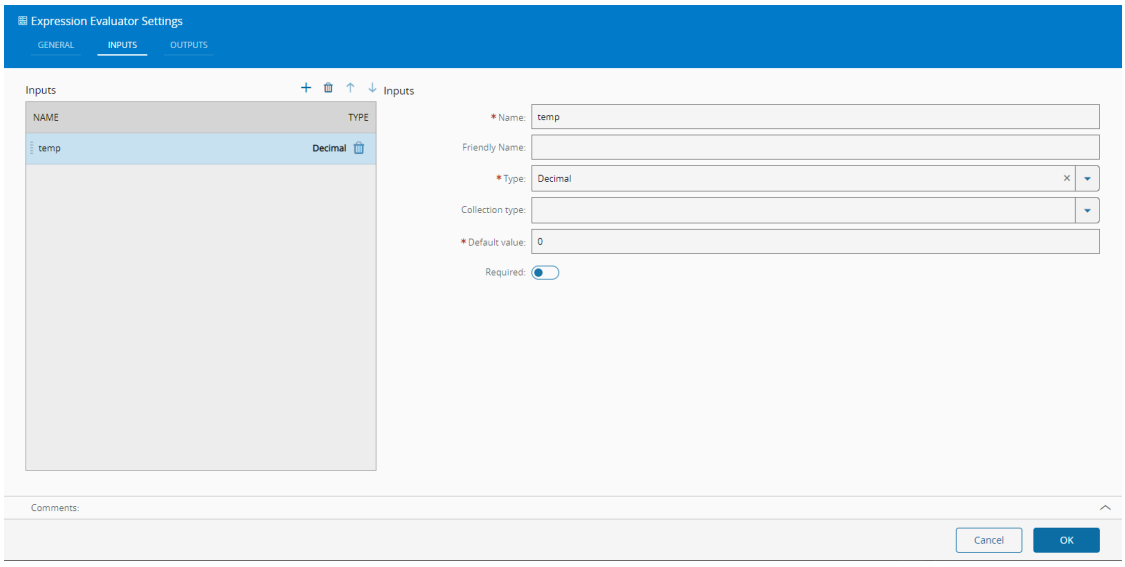
4. Confirm that in the output field the Temperature Property is prompted



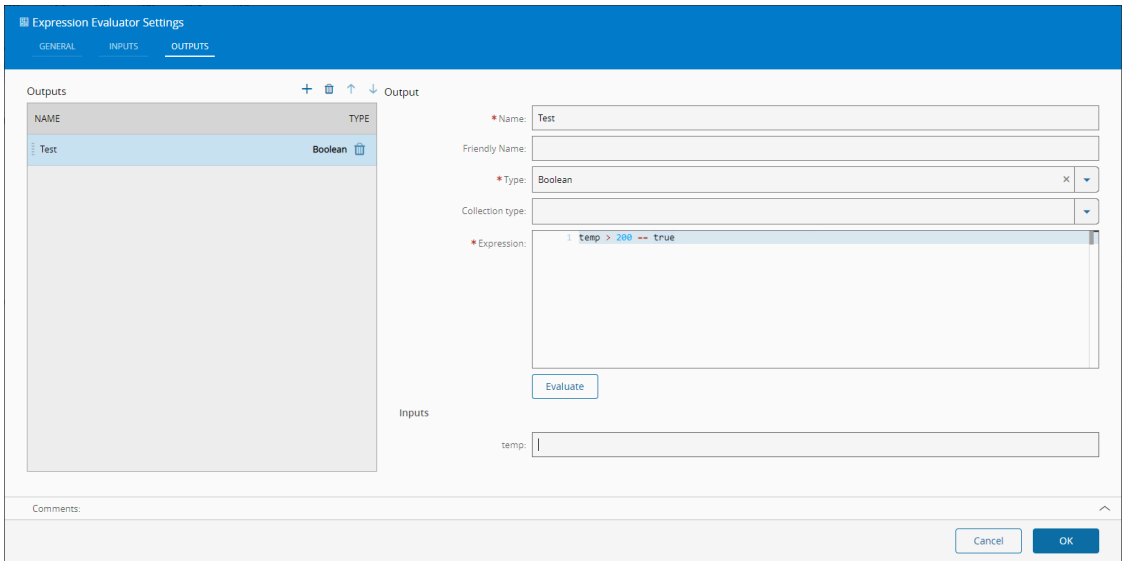
5. Go to the `Expression Evaluator` settings



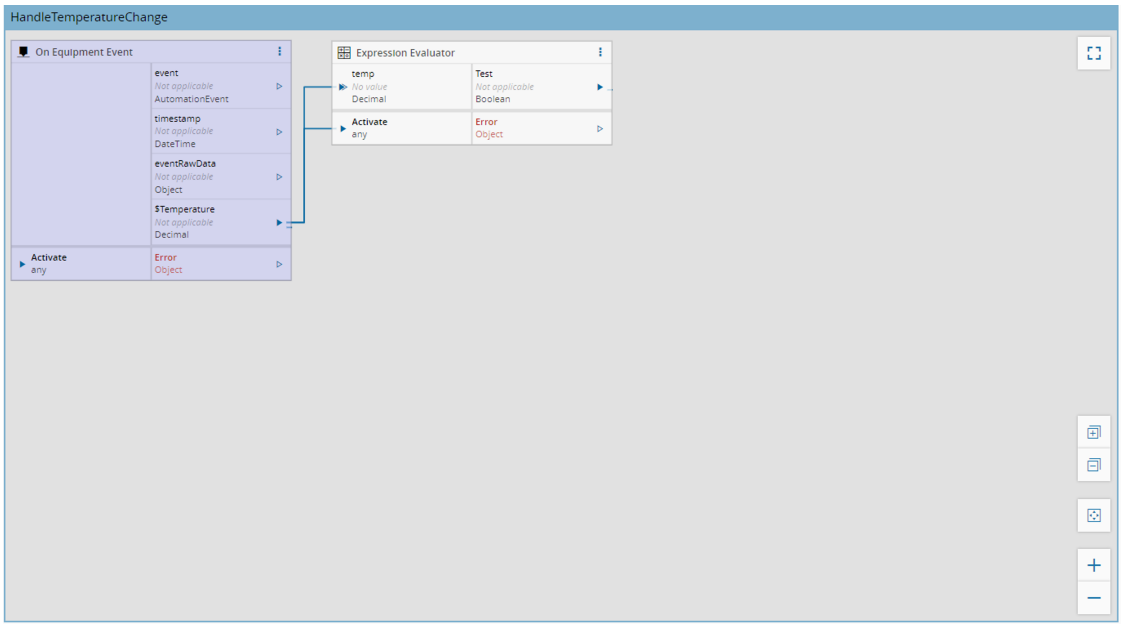
6. Go to the `Inputs` step, select + to add an input and provide:
   - A name
   - A Type
   - A default value



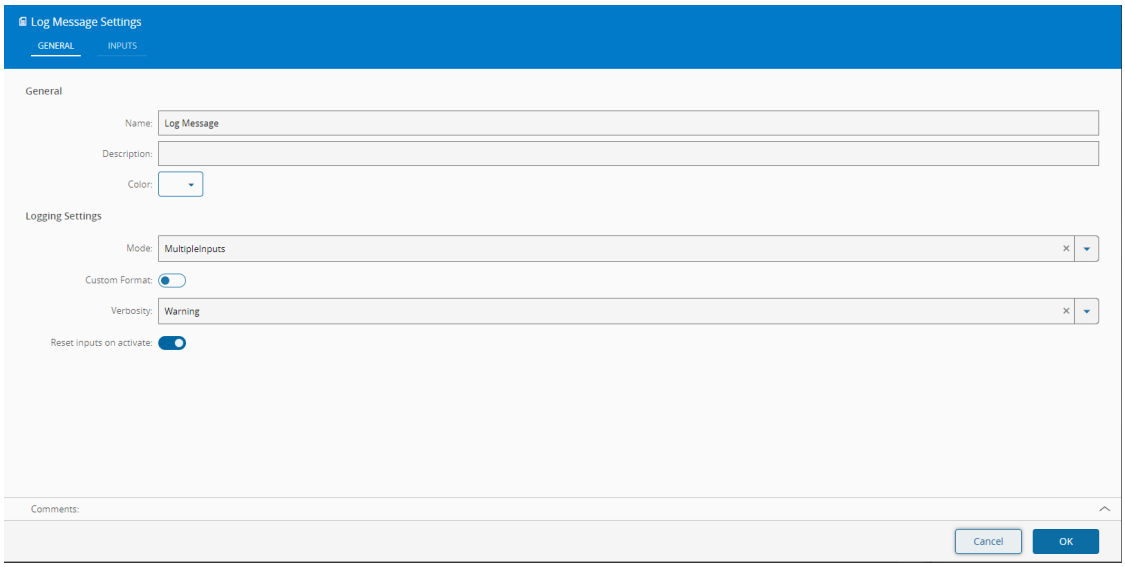
7. Go to the `Outputs` step, select + to add an output and provide:
   - A name
   - A Type
   - The expression to evaluate the Temperature value: `temp > 200 == true`



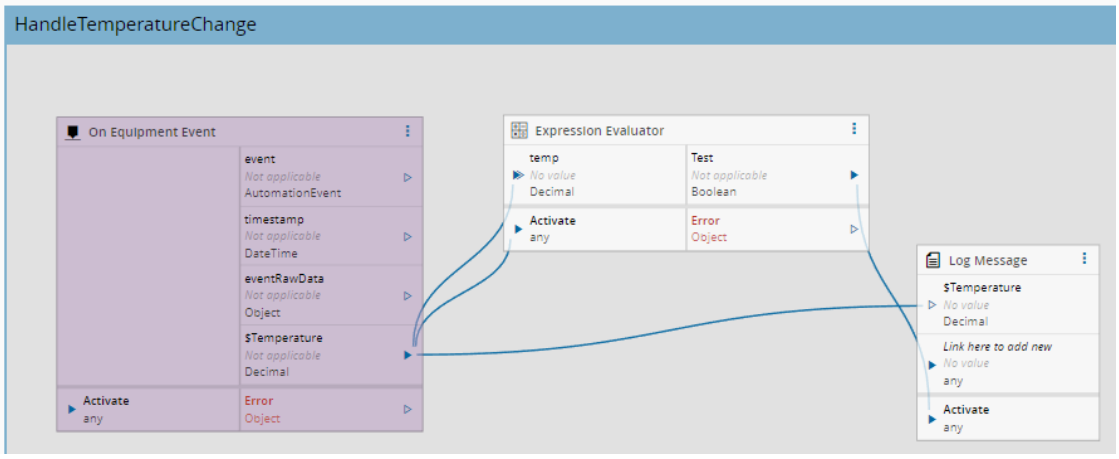
8. Link:
   - The Temperature output of `On Equipment Event` to the `Expression Evaluator` created input
   - The Temperature output of `On Equipment Event` to the `Expression Evaluator` Activate



9. Link the `Expression Evaluator` Output to the `Log Message` Activate
10. Go to the `Log Message` settings, and set the verbosity to warning, select `OK`



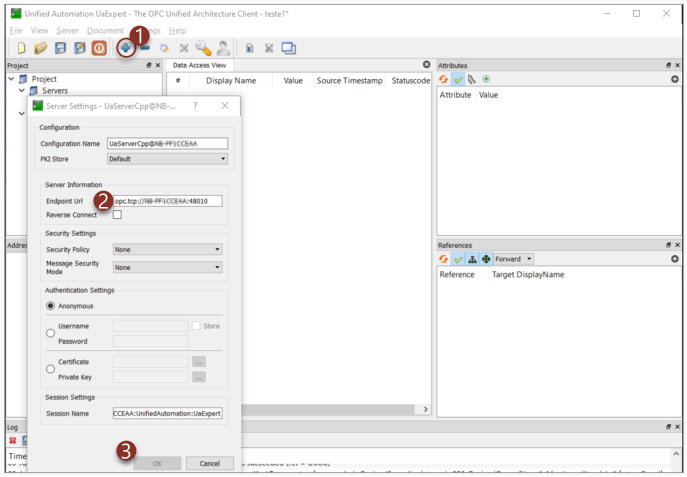
11. Link:
    - The `Expression Evaluator` output `Test` to the `Log Message` Activate
    - The `On Equipment Event` Temperature output to the `Log Message` output



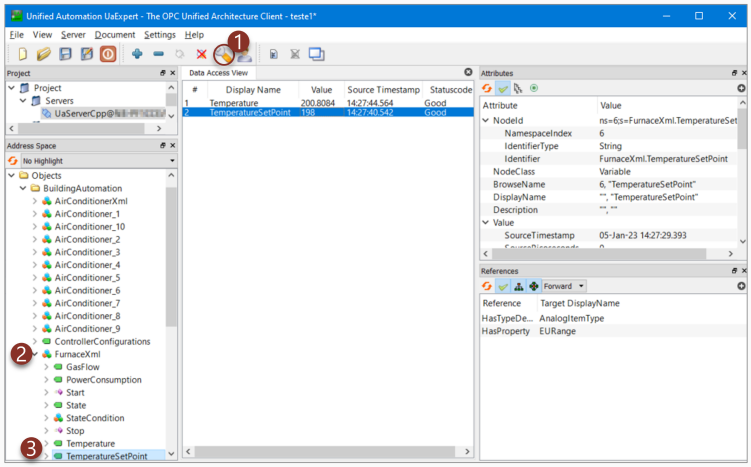
Equipment Simulator Tests

In order to test this integration tutorial, we can use a free OPC/UA server (to mimic the behavior of an equipment) available at https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html as well as an OPC/UA client (to set the values of the server accordingly), available at https://www.unified-automation.com/products/development-tools/uaexpert.html.

1. Go to the websites listed above to download and install the UaCPPServer server software and Unified Automation UaExpert client software.
2. Start the server.
3. Run the UaExpert client. Add a server, set the Endpoint Url to the server displayed on UaCPPServer, and select `OK`



4. Select Connect, and on the address space, go for `BuildingAutomation > FurnaceXml`, and drag and drop Temperature and TemperatureSetPoint to the Data Access View



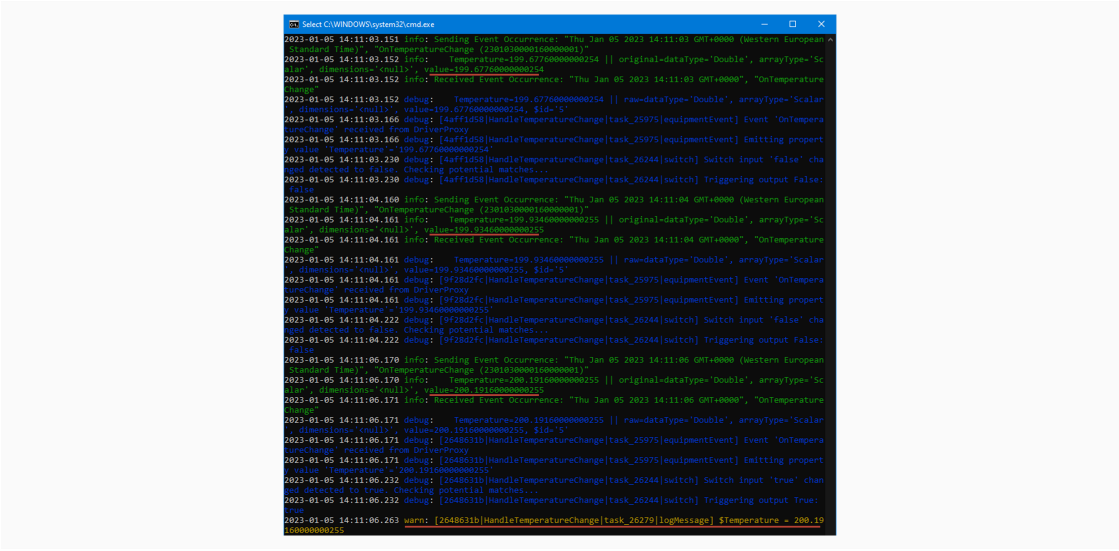
Note

Note that the Automation Manager Console has no warnings logs with the Temperature values.

Note

If you are having issues dragging and dropping the properties to the Data Access View, consider running windows troubleshoot and lowering the Operating System, there have been reported issues for older versions with Windows 11.

1. Set the Temperature Set Point to a value greater that 200. By setting the set point the temperature will rise to match it.
2. The Automation Manager Console will show a warning log entry with the Temperature value.



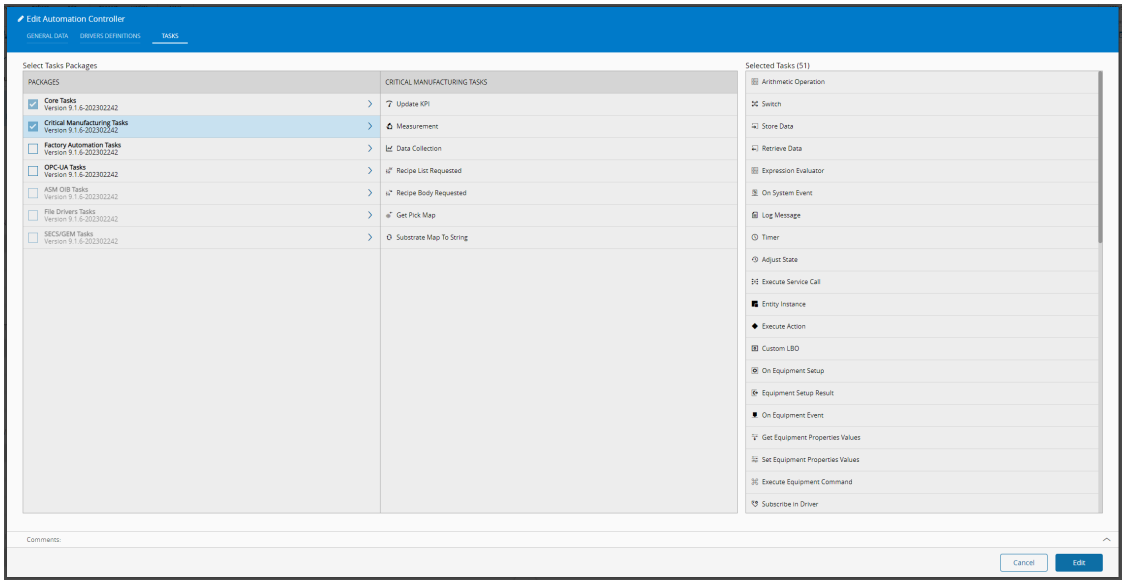
You now have a built structure using Connect IoT that can connect to an equipment and retrieve values according to a specific business logic workflow. This is the end of the intermediate configuration tutorial.

Communicating with the MES

So far we have integrated with the machine, applied conditional logic and now we want to post all the datapoints over the limit on a DataCollection. This could be the use case of the oven not even being operational for temperatures bellow a certain threshold, so we can discard them.

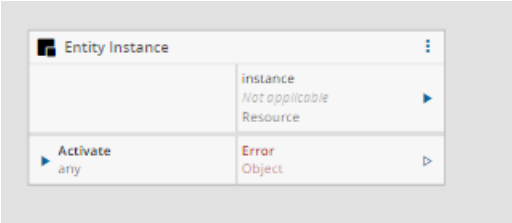
We will now require tasks that are MES specific.

1. In the Automation Controller in the MES, select `Views > Details`, then the `Edit` button and in the Tasks tab and check the `Critical Manufacturing Tasks` and select the button `Edit`. To go back to the workflow select `Views > Workflow`.



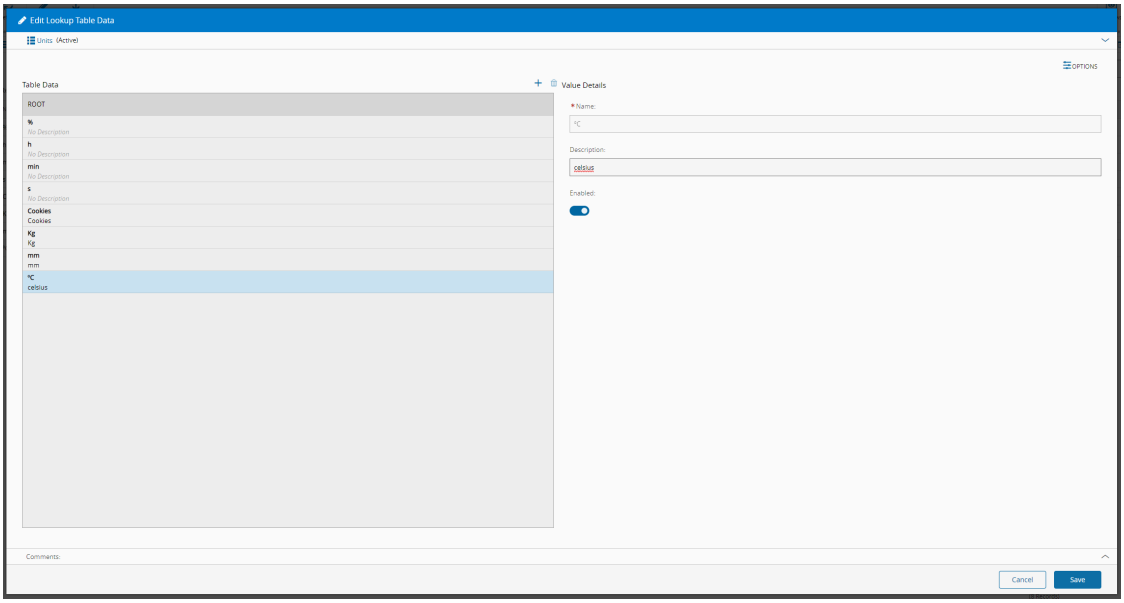
   In the workflow tab, you will now have access to the new tasks. We will use the `Entity Instance` task and the `Data Collection` task.

2. The `Entity Instance` task serves to retrieve the entity that is associated to our Automation Manager, in this case it will be a `Resource` called Baker-01.

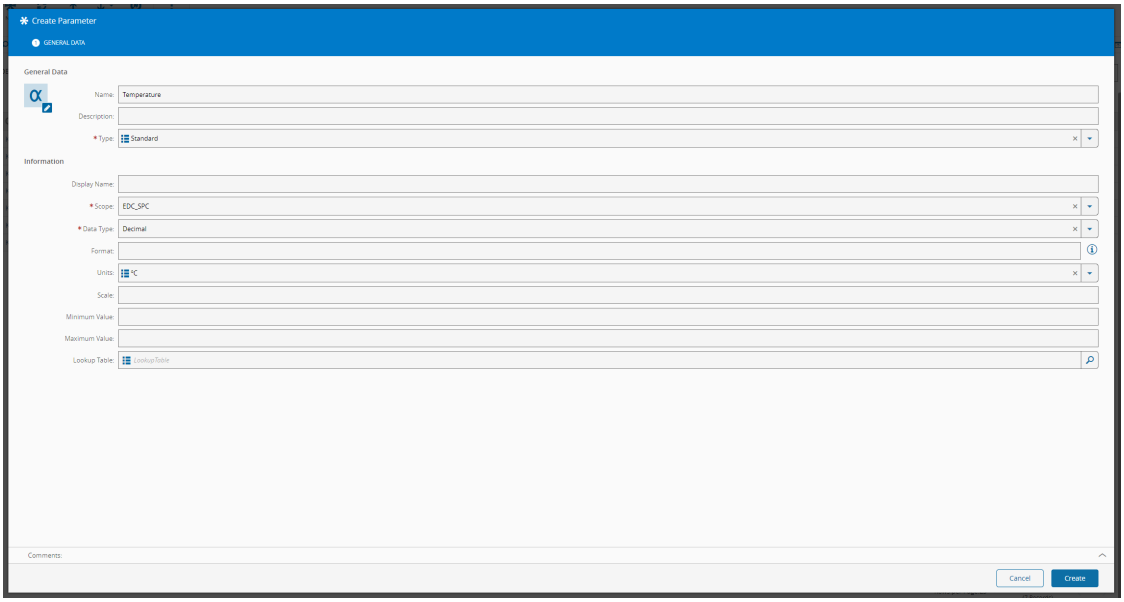


3. Before adding the `Data Collection` task we must create in the MES the Parameter and the Data Collection.

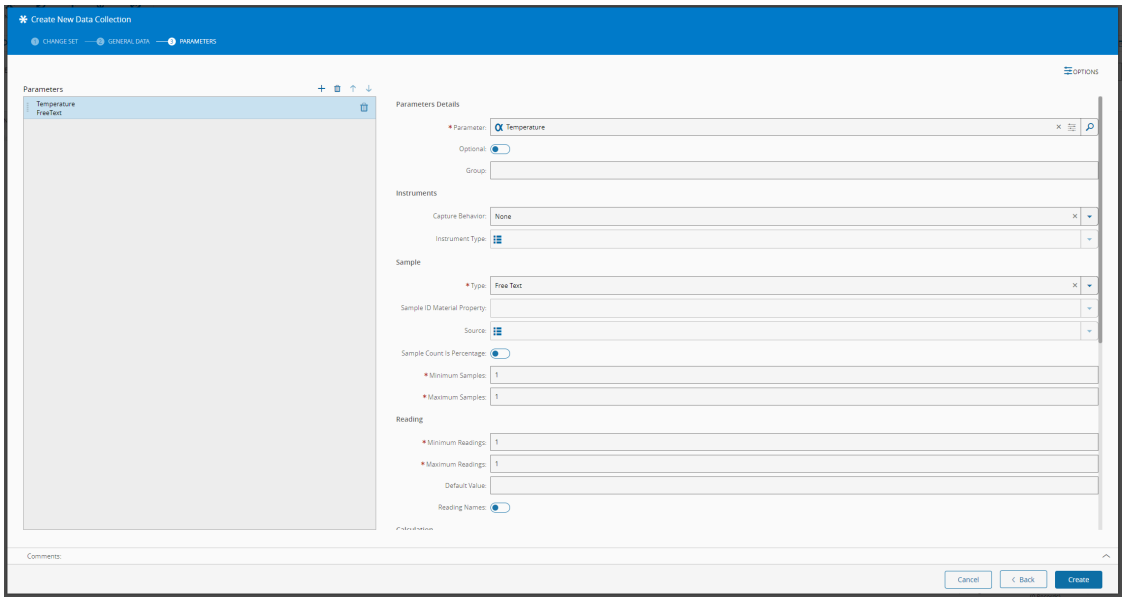
   To do so, first we will need to create a unit, and in order to do that, go to `Administration > Tables > LookUp Tables` and open the `Units` table. Now, let's add a new value to the table, for example ºC and save. Feel free to already add ºK, we will need that later on.



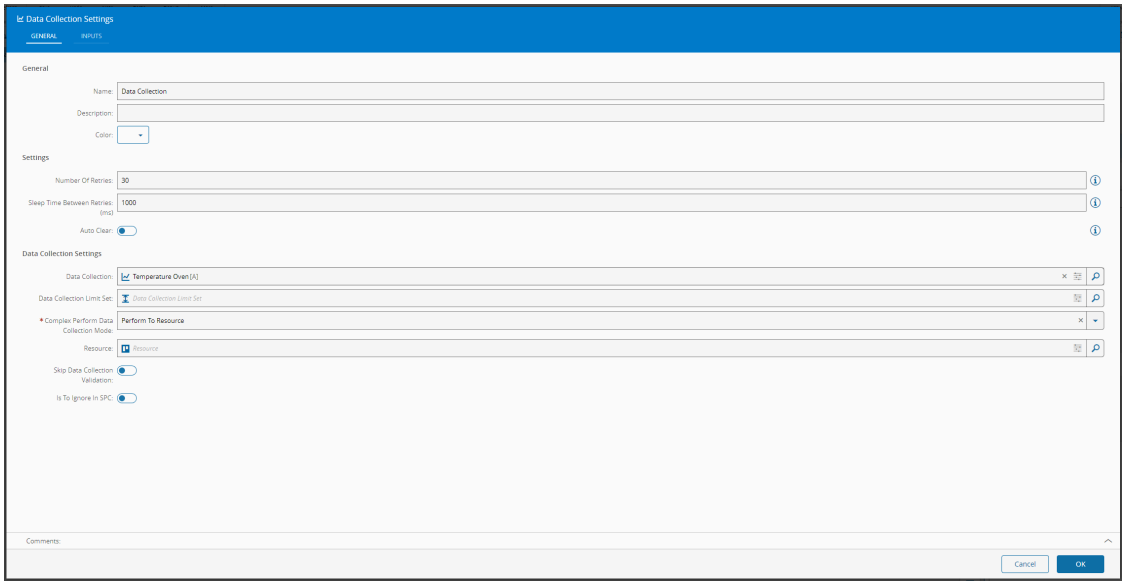
4. Now select `Business Data > Parameter` and select `New`. Create a new parameter with name Temperature, data type Decimal and Units will be ºC and select `Create`.



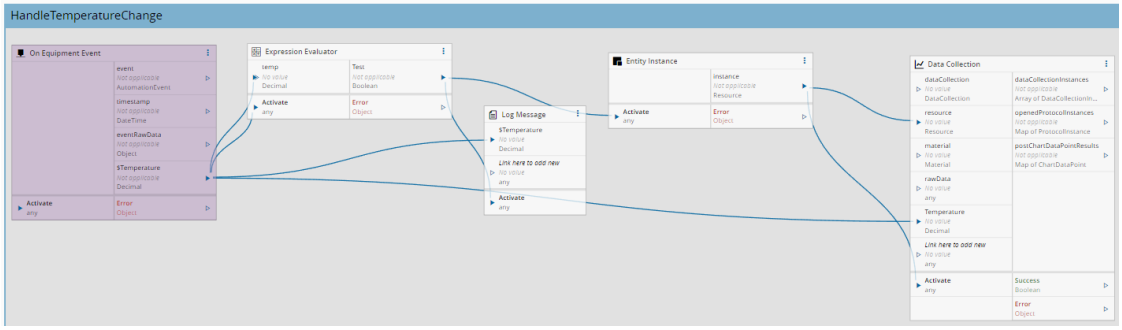
5. We can now create the Data Collection. Go To `Business Data > Data Collection` and select `New`. Leave the defaults, give the name Temperature Oven and use the parameter Temperature we have created.



6. Now add the `Data Collection` task to the Automation Controller workflow. Select the Data Collection Temperature Oven and we will use only the resource scope, so select Perform to Resource, in the Complex Perform Data Collection Mode.



7. Link the `Expression Evaluator` Output to the `Entity Instance` Activate.
8. Link the `Entity Instance` Output to the input resource of the `Data Collection` task and to the `Data Collection` task activate.
9. Link the $Temperature from the `On Equipment Event` to the input Temperature of the `Data Collection` task.



In the console output of the Automation Manager you should now see the posts being performed. Notice that very small changes are generating posts. If you want to see the data in the MES you can go, for example to the Resource Baker-01 and see the Collected Data tab.

The next and final challenge is changing the sensitivity of our post to be just when it changes above 1ºC and then we want to collect it in Kelvins and not in Celsius.

1. If you haven't done so already, add to the Units lookup the ºK value.
2. In the Parameter Temperature, edit the unit to ºK.
3. Delete the `Expression Evaluator` task, it will no longer be needed.
4. In the Automation Controller workflow, we could still use the `Expression Evaluator`, but let's try and use a different task that allows us to do more complex transformations, the `Code` task. This task allows us to write typescript code snippets. The goal for our code task is to handle the transformation to Kelvin, and to check if the temperature is above 200 and if it has 1º of difference to the previous posted value.
   - Link the output `$Temperature` to the `Link here to add new` in the `Code` task and to the Activate of the `Code` task.
   - Add the Output `temperatureKelvin` as type Decimal
   - Replace the code in the main function with the following:

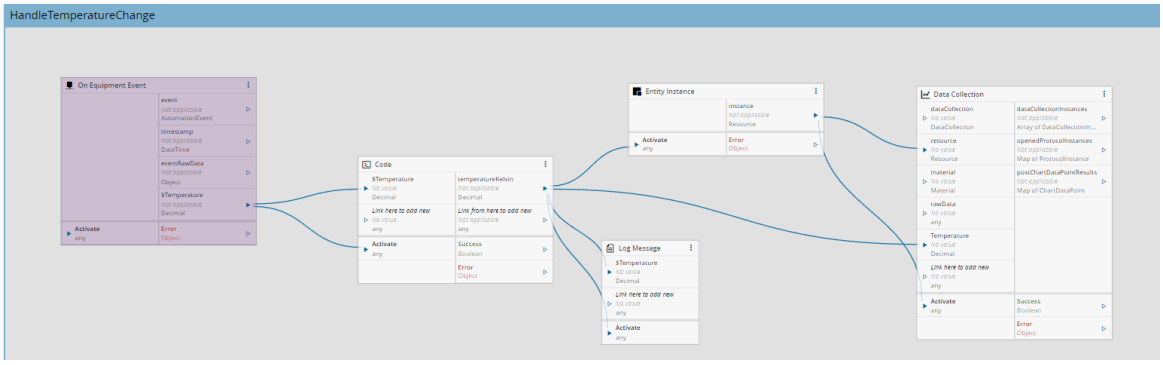
```
    public async main(inputs: any, outputs: any): Promise<any> {

        // Save Input as number
        const temperatureInput = inputs.$Temperature as number;
        // Retrieve last persisted temperature
        const lastTemperature = await this.framework.dataStore.retrieve("lastTemperature", 0);

        // Only check temperatures above 200°C
        if (temperatureInput > 200){
              // Convert to Kelvin
              const temperatureKelvin = temperatureInput + 273.15;
              // Delta of last posted temperature and new temperature
              const temperatureDifferential = Math.abs(temperatureKelvin) -
Math.abs(lastTemperature);

              // Delta must be above 1º
              if (temperatureDifferential > 1) {
                 // Persist new temperature
                 await this.framework.dataStore.store("lastTemperature", temperatureKelvin,
"Temporary");
                 // Emit new temperature
                 outputs.temperatureKelvin.emit(temperatureKelvin);
              }
        } else {
              this.framework.logger.debug("Temperature bellow 200 celsius will be discarded");
        }
    }
```

5. Link the `temperatureKelvin` of the `Code` Task to the input `Temperature` of the `Data Collection` task, to the Activate of the `Entity Instance` task and to the $Temperature and Activate of the `Log Message`.
6. Save the new workflow configuration



In the console output, notice you will now have two new log messages. One for temperature bellow 200ºC:

```
2023-03-20 14:00:40.941 info: Sending Event Occurrence: 'Mon Mar 20 2023 14:00:40 GMT+0000 (Western European Standard Time)', 'OnTemperatureChange (230315103056000000l)'
   Temperature=199.95739999998423 || original=dataType='Double', arrayType='Scalar', dimensions='<null>', value=199.95739999998423
2023-03-20 14:00:40.941 info: Received Event Occurrence: 'Mon Mar 20 2023 14:00:40 GMT+0000', 'OnTemperatureChange':
   Temperature=199.95739999998423 || raw=dataType='Double', arrayType='Scalar', dimensions='<null>', value=199.95739999998423, $id='5'
2023-03-20 14:00:40.942 debug: [35a6033b|HandleTemperatureChange|task_7814|equipmentEvent] Event 'OnTemperatureChange' received from DriverProxy
2023-03-20 14:00:40.942 debug: [35a6033b|HandleTemperatureChange|task_7814|equipmentEvent] Emitting property value 'Temperature'='199.95739999998423'
2023-03-20 14:00:40.972 debug: Retrieving data identified with 'lastTemperature'
2023-03-20 14:00:40.973 debug: [<<root>>|HandleTemperatureChange|task_2298|codeExecution] Temperature bellow 200 celsius will be discarded
```

Another message for, when the temperature differential is above 1ºC:

```
2023-03-20 13:58:29.907 info: Sending Event Occurrence: 'Mon Mar 20 2023 13:58:29 GMT+0000 (Western European Standard Time)', 'OnTemperatureChange (230315103056000000l)'
   Temperature=201.33999999998431 || original=dataType='Double', arrayType='Scalar', dimensions='<null>', value=201.33999999998431
2023-03-20 13:58:29.908 info: Received Event Occurrence: 'Mon Mar 20 2023 13:58:29 GMT+0000', 'OnTemperatureChange':
   Temperature=201.33999999998431 || raw=dataType='Double', arrayType='Scalar', dimensions='<null>', value=201.33999999998431, $id='5'
2023-03-20 13:58:29.908 debug: [f43afa34|HandleTemperatureChange|task_7814|equipmentEvent] Event 'OnTemperatureChange' received from DriverProxy
2023-03-20 13:58:29.908 debug: [f43afa34|HandleTemperatureChange|task_7814|equipmentEvent] Emitting property value 'Temperature'='201.33999999998431'
2023-03-20 13:58:29.939 debug: Retrieving data identified with 'lastTemperature'
2023-03-20 13:58:29.939 debug: Storing data in 'Temporary' for 'lastTemperature': 474.4899999999843
2023-03-20 13:58:29.955 warn: [f43afa34|HandleTemperatureChange|task_2444|logMessage] *** 474.4899999999843 in Kelvins ***
2023-03-20 13:58:29.986 debug: [f43afa34|HandleTemperatureChange|task_10931|dataCollection] Performing data [$id='<null>', $type='Cmf.Navigo.BusinessObjects.DataCollectionPoint, Cmf.Navigo.BusinessObjects', TargetEntity={ob
ject: $id='<null>', $type='Cmf.Navigo.BusinessObjects.Parameter, Cmf.Navigo.BusinessObjects', Name='Temperature'}, Value=474.4899999999843, ReadingNumber=1] into Resource Baker-01...
2023-03-20 13:58:30.170 info: [f43afa34|HandleTemperatureChange|task_10931|dataCollection] Successfully performed data into Resource Baker-01.
```

You now have a built structure using Connect IoT that can connect to an equipment and retrieve values according to a specific business logic workflow. This is the end of the intermediate configuration tutorial.

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.