Critical manufacturing
an ASM PT company

Advanced

11.0

March 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.

Connect IoT - Advanced Configuration Tutorial

Estimated time to read: 32 minutes

This tutorial builds upon the (Basic and Intermediate) 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, in the intermediate we posted to a Data Collection.

In the `Advanced Tutorial`, we will first make the integration configurable in terms of connection settings, then we will address the scenario where we have a setup of a resource where we set the setpoint to 200ºC and validate it. When the setpoint reaches the 200ºC the user will be able to perform a Track In and post the temperature to a Data Collection. After 1 minute it will perform the Track Out of the Material. An additional requirement is the ability to collect temperature values on demand.

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

MES Model

Let's start with building the model for this tutorial:

1. Create a Calendar
2. Create a Facility
3. Create an Area
4. Create a Resource named `Oven`
5. Create a Step named `Oven Step`
6. Create a Service `Oven Service`
    a. Add the service to the Resource Services
    b. Add the service to the Step Context
7. Create a Flow named `Oven Flow` with the `Oven Step`
8. Create a Product named `Product Oven`
    a. Give as default flow path the Flow `Oven Flow` and the Step `Oven Step`
9. Create a Material `Oven Material`
10. Dispatch the `Oven Material` to the Resource `Oven`

If the `Oven Material` is not dispatchable to the Resource `Oven`, please revalidate your model.

> **Note**
>
> You can always change the model to what best suits you. This tutorial is not focused in MES modeling so it will be very brief on this topic. Also, feel free to use the model you have from previous tutorials and add the parts that are missing from your model.

Automation Driver Definition

Let's go over the **Automation Driver Definition** `Oven DD` and set the properties. In this case we will not need events, only properties and commands.

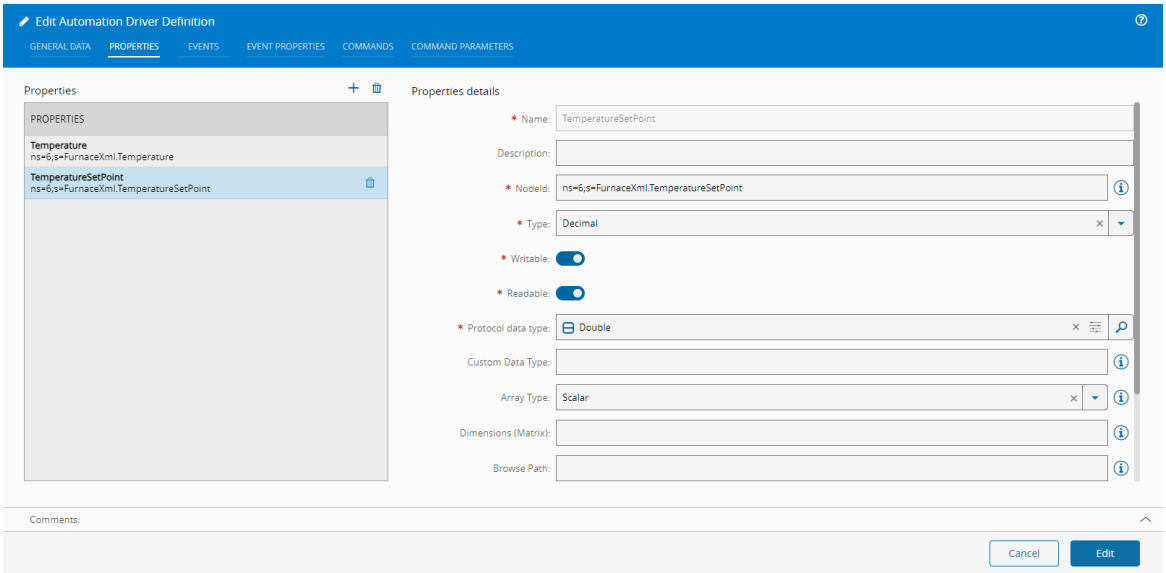
> **Note**
>
> Note that to do this in a real life context, it is important to have a general knowledge about the protocol, the equipment itself and its related documentation.

Properties

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

To add the `TemperatureSetPoint` 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



In this case we have no commands or events. If we chose to add any command of event, they would be defined in the appropriate tabs.

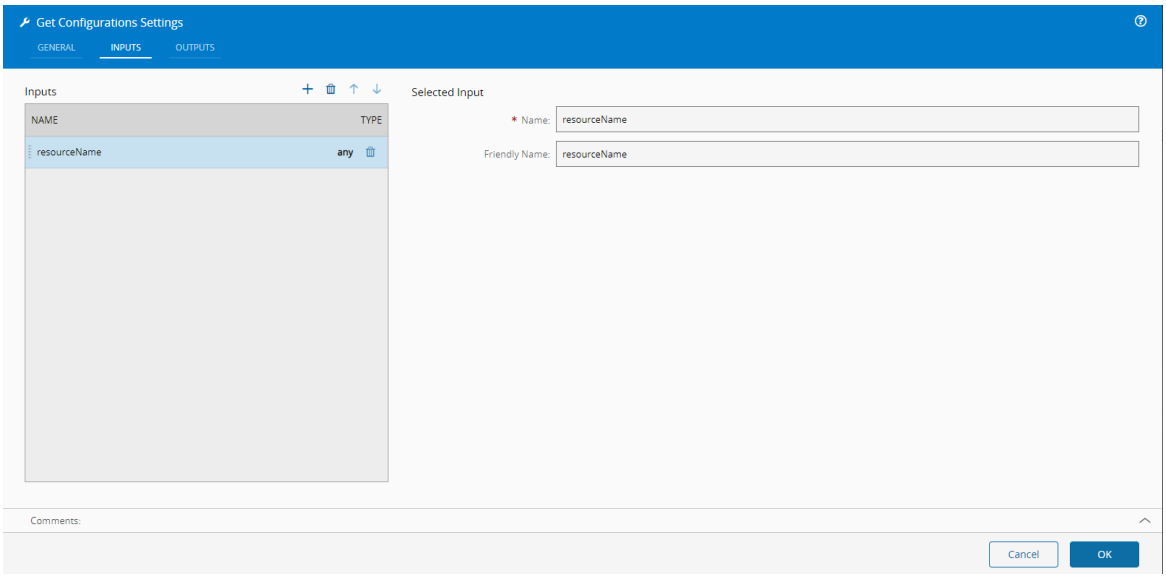
Automation Controller

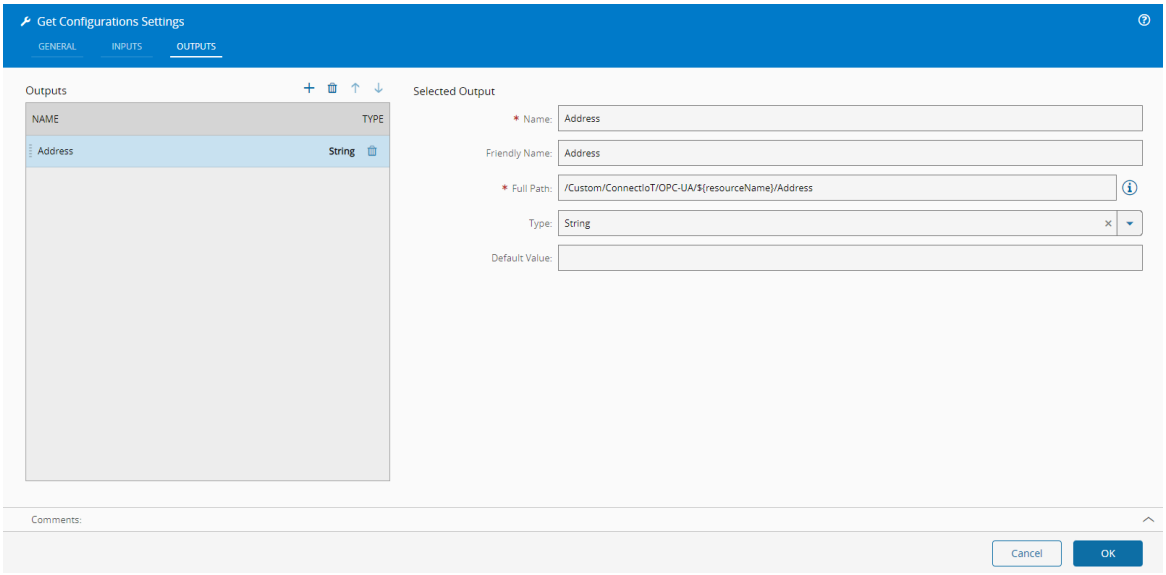
Let's go over the **Automation Controller** `Oven Controller` and define the logic that will support the described scenario.

Setup

First, let's review how we are addressing the setup. Currently, we have the values statically defined in the workflow, but in order to reuse the same controller for different entities, we need the workflow to be a bit more dynamic. Let's start with the following steps:

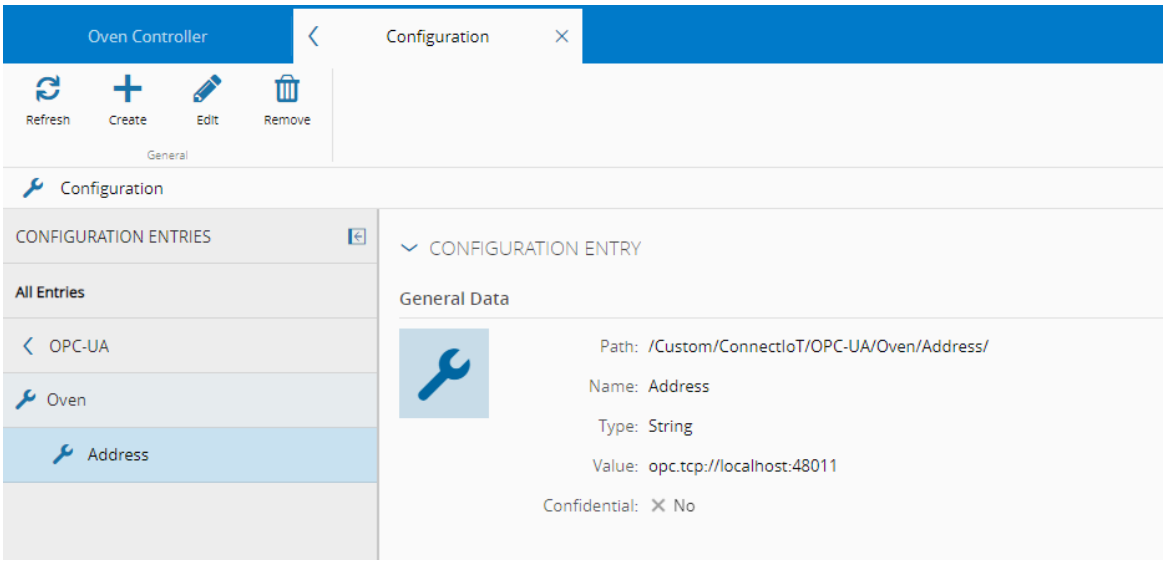
1. Drag and drop the following tasks:
   - Get Configurations: to retrieve Configurations from the system (`Administration > Configurations`)
   - Entity Instance: to assess the associated entity
2. Connect the output of `OnSetup` of the `On Equipment Setup` task to the `Entity Instance` active
3. Remove the link of `OnInitialize` to the `connect` link
4. In the `Get Configurations` task:
   a. Create input `resourceName` - Inputs in the `Get Configurations` can be used as tokens
   b. Create output `Address`
   c. Give as path `/Custom/ConnectIoT/OPC-UA/${resourceName}/Address` - notice `${resourceName}` will be replaced by the defined input
   d. Type as `String`





1. Link the `Entity Instance` instance to the activate of the `Get Configurations`
2. Link the `Entity Instance` instance to the resource name input of the `Get Configurations`
3. The instance comes with the full object payload and we just need the `Name`
    a. Create converter `AnyToAny`
    b. Create converter `GetObjectProperty`
        i. With path `Name`
        ii. Type `String`
4. Link the `Get Configurations` success to connect of the `On Equipment Setup` task
5. Go to `Administration > Configuration`
6. Click `Create`
7. Create configuration entries until you have the desired path of `/Custom/ConnectIoT/OPC-UA/Oven`
8. Create the config that will hold the value
    a. In the `Name` write `Address`
    b. In the `Type` select `String`
    c. Set the value with the proper address (i.e `opc.tcp://localhost:48101`)

9. Link the output `Address` of the `Get Configurations` to the `OnEquipmentSetup` `Address` input



Now, the hook to determine the address will no longer be the `Automation Controller` but rather the configuration for a particular resource in the configurations entry. This is a simple way to make our controller fully dynamic and in that way promote its reusability throughout different resources of the same type. There are other mechanisms to hold these values such as saving in `Entity Attributes` or on a separate table, both of these are also out of the box supported by tasks, with the `Entity Instance` task and the `Resolve Table` task.

On Demand - Show Temperature

One of the interesting applications of Connect IoT is providing direct feedback to a GUI, in order to have a more powerful and symbiotic integration between the equipment and the operator. The goal of our GUI is to be able to retrieve and display the latest temperature value. This is a very simple example, but it will provide the basic building blocks of the system.

Create a DEE - GetCurrentTemperature

Let's create a DEE to retrieve the Temperature from Connect IoT.

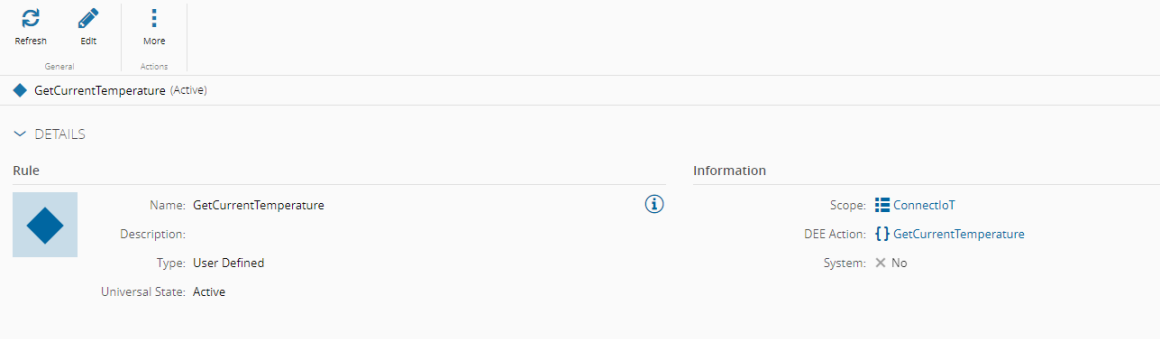
1. Go to `Administration > DEE Actions`
2. Select `New`
3. Give as Name `GetCurrentTemperature`
4. Give Classification `ConnectIoT`
5. Use the following code:

```
// Retrieve Service Provider
var serviceProvider = (IServiceProvider)Input["ServiceProvider"];

IResource resource = serviceProvider.GetService<IResource>();

// Retrieve Resource Name from the Inputs
resource.Name = Input["Resource"] as string;
resource.Load();

var instance = resource.GetAutomationControllerInstance();

// Validate there's an Automation Controller instance for this resource
if (instance == null) {
    throw new Exception("Resource not connected to any IoT instance");
```

```
    }
    else
    {
        // Request value from Connect IoT and wait for the reply
        dynamic payload = instance.SendRequest("Cmf.Request.Temperature", null, 10000);

        Input.Add("Value", payload["reply"].ToString());
    }
```

In the DEE execution, it will receive a resource name, resolve the instance linked to the resource and send a message to Connect IoT and wait for a reply.

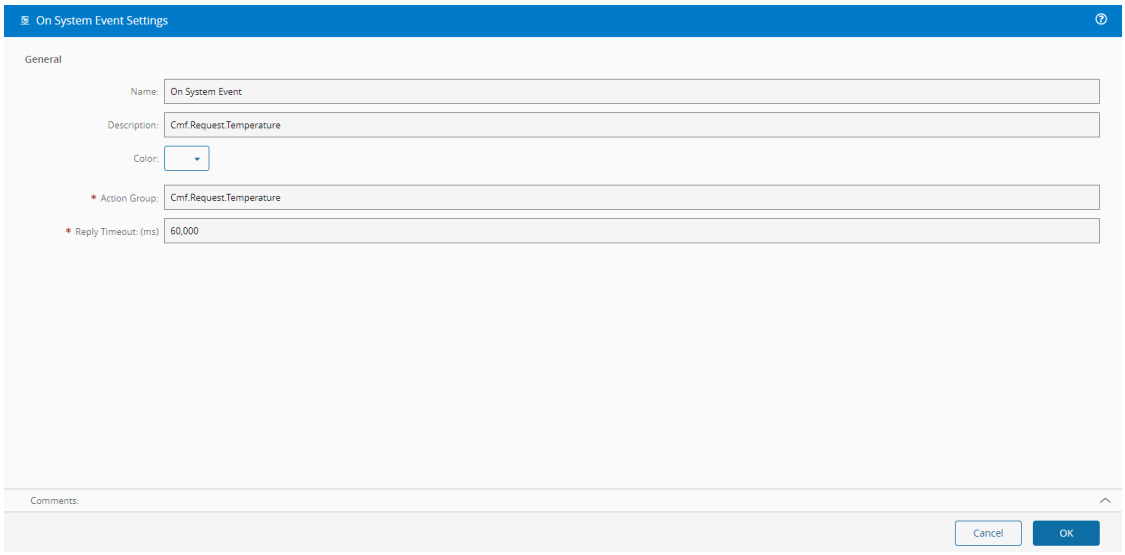
1. Go to `BusinessData > Rule`
2. Select `New`
3. Give as Name `GetCurrentTemperature`
4. Give as Scope `ConnectIoT`
5. DEE Action `GetCurrentTemperature` - if it does not appear in the search box, validate that the DEE Classification is correct



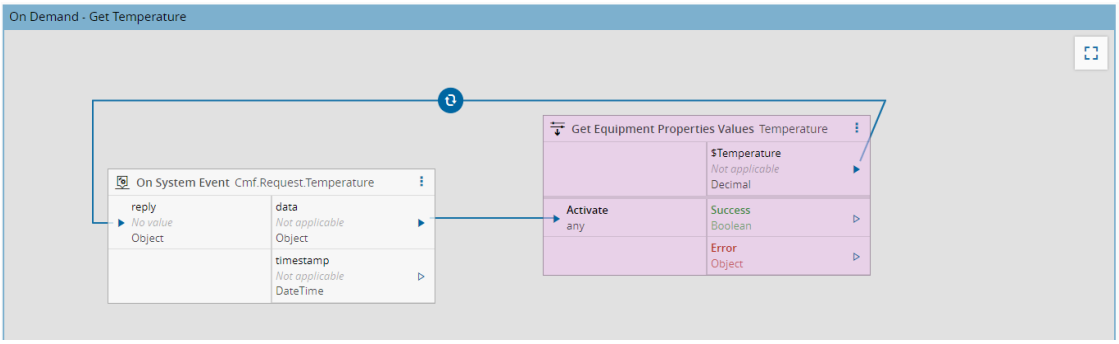
Automation Controller - Retrieve Temperature

In the Automation Controller it should now, upon the message received from the DEE, retrieve and reply back with the temperature value.

1. Go to `Oven Controller`
2. In the Workflow, create a new page `On Demand - Get Temperature`
3. Drag and drop the following tasks:
   - On System Event: to subscribe to the message bus topic `Cmf.Request.Temperature`
   - Get Equipment Properties Values: to get the value of the temperature
4. Go to the `On System Event` settings and for the Action Group, write `Cmf.Request.Temperature`.



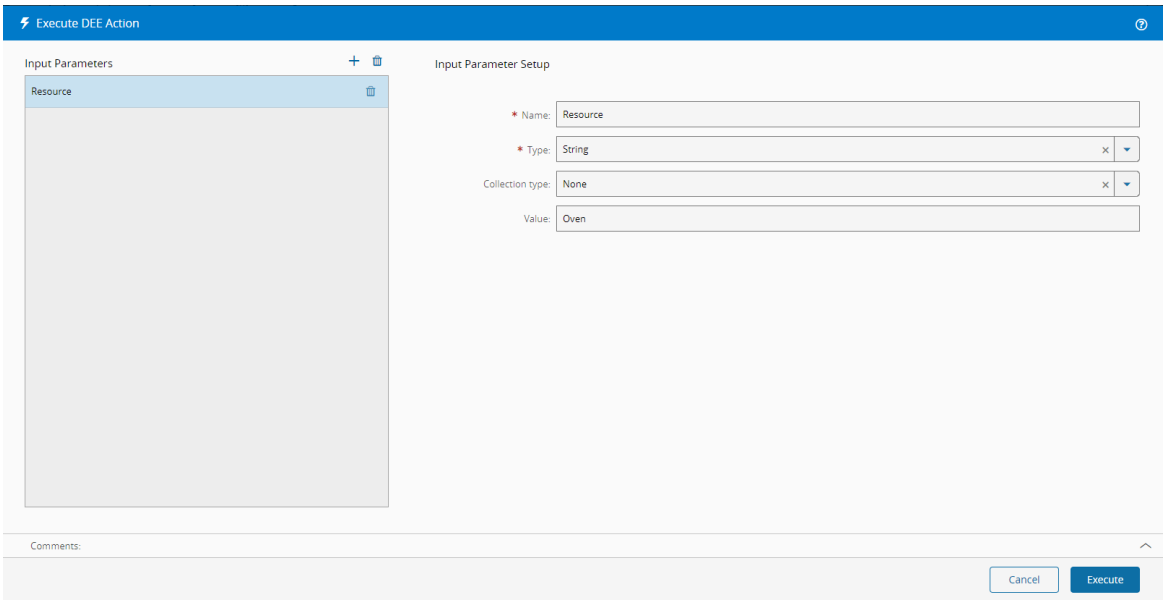
5. In the `Get Equipment Properties Values` task
    a. Add as Output, the Automation Property `Temperature`
6. Link the `On System Event` output `data` to the `Get Equipment Properties Values` Activate input
7. Link the `Get Equipment Properties Values` `Temperature` output to the input `reply`
8. Add a converter `AnyToAny`

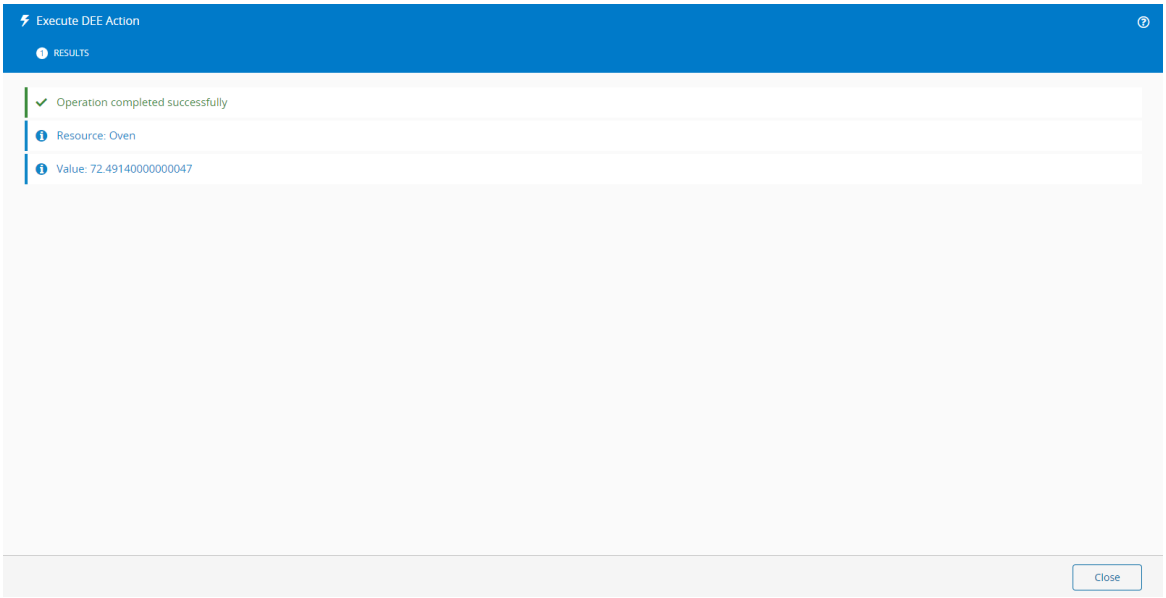


Test with the DEE

Start the OPC UA Server mentioned in the previous (Basic and Intermediate) tutorials and start your Automation Manager, making sure you have the correct address configured in the configuration entry for your resource, which in this case should follow along the lines of `/Custom/ConnectIoT/OPC-UA/Oven/Address`. Go to the DEE that we have created (`GetCurrentTemperature`) and select the Execute button. You can now perform executions of the DEE.

Let's add an input `Resource` with Value `Oven` (or whatever Resource you have linked to the controller instance). Select `Execute`.





Notice how we can easily test our implementation with DEEs. Now let's create a GUI.

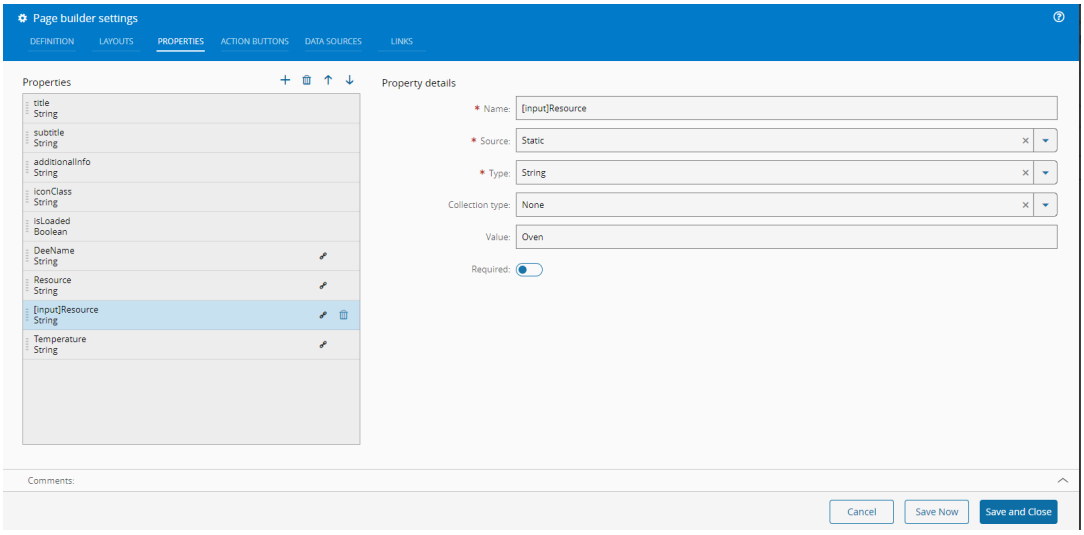
Create a GUI - Retrieve and Show Temperature

MES supports an out of the box approach to create UI pages directly, in the system, without the need to code.

1. Go to `Administration > UIPages`
2. Select `New`
    a. Give the Name `Collect Temperatures`
    b. Press `Create`
3. Select `Edit`
4. Drag and drop the following Widgets:
    - `Form`: to be able to select a Resource from the system
    - `Button`: to trigger the action of getting the temperature
        - Feel free to go to the settings and give it a friendly name like `Get Current Temperature`

- `Text`: to trigger the action of getting the temperature
  - Feel free to go to the settings and give it a friendly name like `Current Temperature`

5. In the `Form` in `Settings > Fields`
6. Add a field `Resource`
   a. Type - `ReferenceType`
   b. Reference type - `Entity`
   c. Reference type name - `Resource`
7. In the `Settings`
   a. In `Properties`, we will define the static variables global variables that we will need
      i. Add a new property for the GUI to know the DEE to execute
         A. Name - `DeeName`
         B. Source - `Static`
         C. Type - `String`
         D. Value - `GetCurrentTemperature`
      ii. Add a new property for the GUI to know the input of the DEE
         A. Name - `Resource`
         B. Source - `Static`
         C. Type - `String`
         D. Value - `Resource`
      iii. Add a new property for the GUI to know the value of the input of the DEE
         A. Name - `[input]Resource`
         B. Source - `Static`
         C. Type - `String`
         D. Value - `Oven` (this is just a default value)
      iv. Add a new property for the GUI to know the value of the output of the DEE with the temperature
         A. Name - `Temperature`
         B. Source - `Static`
         C. Type - `String`
         D. Value - `Value` (this matches the input we added to the DEE `GetCurrentTemperature`)



b. In `Data Sources`, we will define the services we need to call to execute the DEE

i. Add a new datasource for the GUI to Load the DEE to execute

A. Name - `LoadDEE`

B. Type - `ServiceCall`

C. Retrieve data on start - `false`

D. Retrieve data on changes - `false`

E. Select `Settings`

I. Choose `DynamicExecutionEngine`

I. Select `GetActionByName`

ii. Add a new datasource for the GUI to execute the DEE

A. Name - `Request Temperature`

B. Type - `ServiceCall`

C. Retrieve data on start - `false`

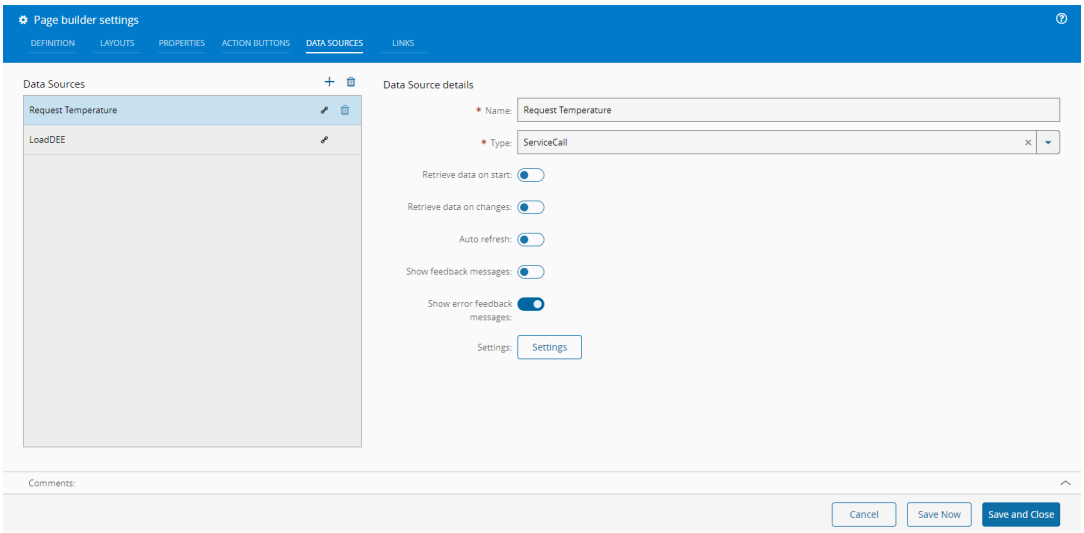
D. Retrieve data on changes - `false`

E. Show error feedback messages - `true`

F. Select `Settings`

I. Choose `DynamicExecutionEngine`

I. Select `ExecuteAction`



c. Press `Save and Close`

8. In the right pane, select `Links`

a. Drag and drop the `Form` widget

b. Link the `Form` output `field$ResourceChange` to the `Page` input `[input]Resource`

i. Add the converter `entityName`. This will retrieve the Resource name.

c. Drag and drop the `LoadDee`

d. Link the output of `Page` `DeeName` to the input of `LoadDee` `Name`

e. Drag and drop the button `GetCurrentTemperature`

f. Link the button `GetCurrentTemperature` output `OnButtonClick` to the input `refresh` of the `LoadDee`. This means that every time you press the button it will refresh this widget

g. Drag and drop the widget `Request Temperature`

h. Connect the output of the `Page` `[input]Resource` to the Input `Input` of the `Request Temperature`

   i. Apply the converter `setMapValue` with converter parameter `Resource`. This will associate the value of the property resource, which in this case is the constant string `Resource` with the value of the `[input]Resource`. Creating a Map of key `Resource` and value, the value that was fed to the `[input]Resource`, in this case the actual resource name.

i. Link the output `output$Action` of the `LoadDee` to

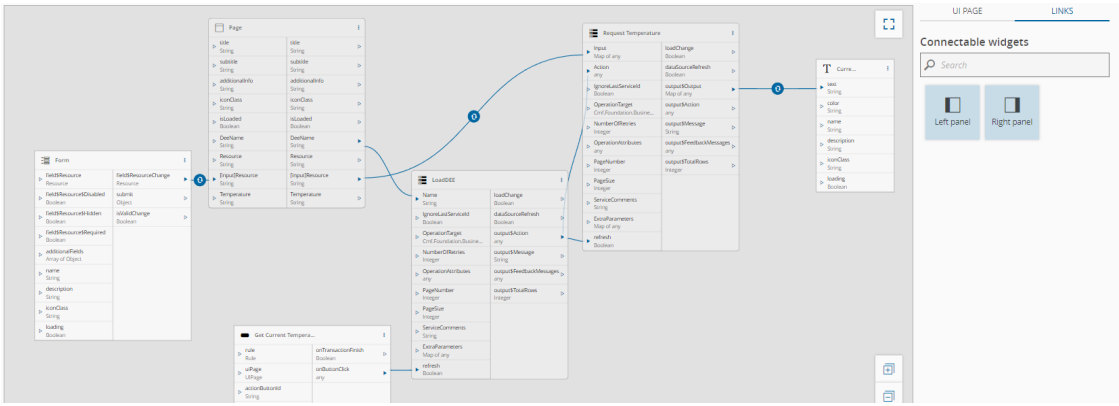
   i. `Action` input of the `Request Temperature`

   ii. `refresh` input of the `Request Temperature`

j. Drag and drop the widget `Text`

k. Link the `output$Output` of the `Request Temperature` to the input `text` of the `Text` widget

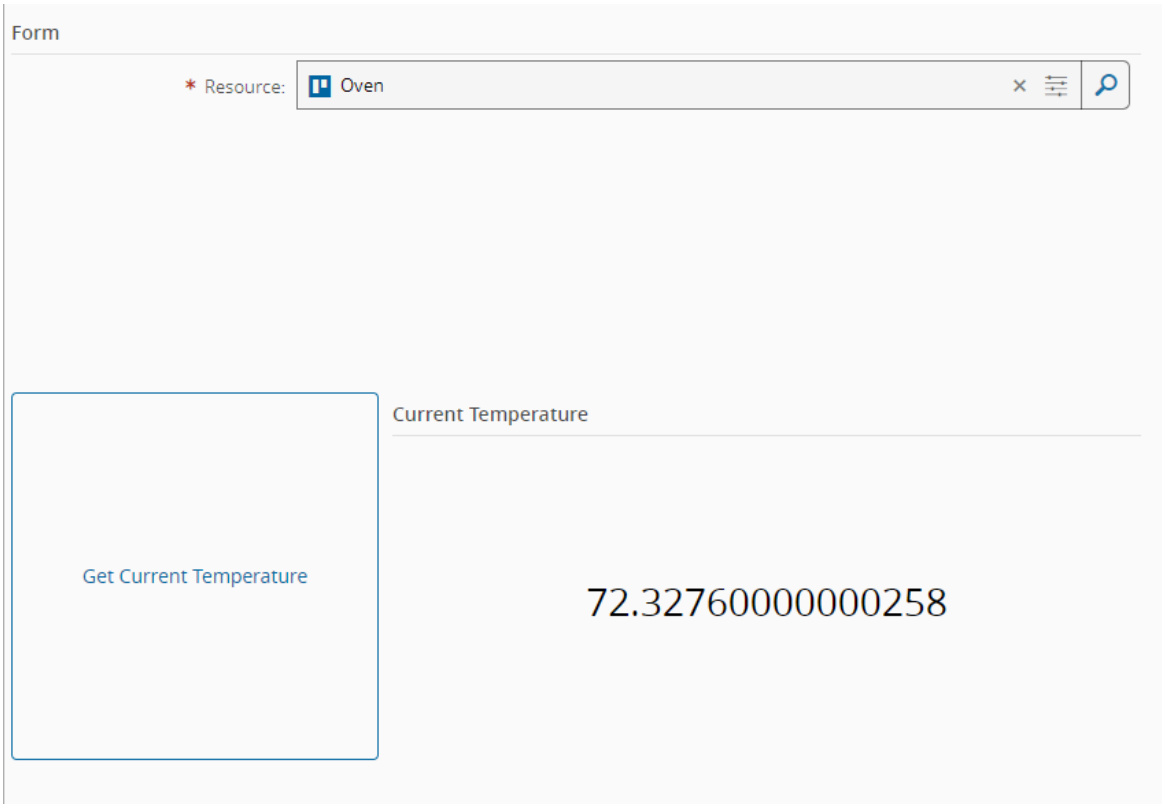
l. Apply the converter `anyToStringProperty` with converter parameter `Temperature`. This will look into the output map for a key of the value that is declared in the property `Temperature`, in this case is `Value`.



9. Press `Save`

The page is now fully usable. Provide a resource to the form that does not have a controller instance you should see an exception popup as a banner.

If we select the `Oven` resource and press the button `Get Current Temperature` we will see now the temperature is show in the text box.



Note

If the DEE has multiple Inputs, the methodology is the same but with multiple links to the input `Input`.

Set Setpoint - On Resource Begin Setup

The use case can be further built upon if we consider that this value was set externally (e.g. through a recipe). Right now the focus is just to show the interaction between the system and Connect IoT. The goal is to have a Setup for a Resource in the MES that can only be complete if the Setpoint is set to 200ºC.

We will create a DEE with the value 200ºC statically defined that will send a message to IoT and await confirmation in the Resource Complete Setup action. This value could be changed to come from a recipe, a definition in a table, a configuration or some other definition in the system in order to have the implementation become dynamic.

Create a DEE - SetSetpointTo200

In order to be able to request an action to be performed appended to a system interaction we will need to create a hook on the action of the Begin Setup. The way to implement this in the MES system is through the use of DEEs that can be appended in the system actions either in `Pre` (before execution), or `Post` (after execution). These actions are within the transaction of the action performed, so if they give an error the whole transaction will rollback and maintain consistency in the system.

1. Go to `Administration > DEE Actions`
2. Select `New`
3. Give as Name `SetSetpointTo200`, for now the classification and action group is not important

The DEE code is split between two important sections. The condition phase and the execution phase. Only if the condition phase returns with true, will the execution phase be invoked. In the history of an action in

the MES this will also be explicit, if a DEE is appended. For more information on DEEs, see DEE Actions.

The context of the DEE, that is present on the Inputs dictionary, will depend on where the DEE is hooked.

Let's add the Action Group to our DEE.

1. Go to the `Details` tab
2. Press `Add` on the Action Group
3. Search for `ResourceManagement.ResourceManagementOrchestration.BeginSetup.Post`
4. Check it and press `Add`

If in step 3. the action group is not present:

1. Go to `Administration > DEE Actions`
2. Select the Settings (three vertical dots) next to the `Action Groups`
3. Select `Add new Action Group`
4. Give as `Name` - `ResourceManagement.ResourceManagementOrchestration.BeginSetup.Post`
5. Select `Create`

In order to find the action group where we want to append our DEE, you can consult the API documentation, or analyze the history whenever the action that you are interested is executed and it will be apparent what are multiple sub-actions that you can append your business logic. Note also that in the DEE in the code view, in the right pane it `Input Parameters`, it will now show all available parameters.

Starting on the code for the `Test Condition Code`. We want to validate that the Inputs that we are interested are correct, to validate we should process and then pass that value to our DEE context.

```
    /// <summary>
    /// Summary text: Send Setpoint to IoT on Begin Setup
    /// Actions groups:
    ///     * ResourceManagement.ResourceManagementOrchestration.BeginSetup.Post
    /// Depends On:
    /// Is Dependency For:
    /// Exceptions:
    /// </summary>

    var serviceProvider = (IServiceProvider)Input["ServiceProvider"];

    // Validate input
    if (Input["BeginSetupInput"] is not BeginSetupInput beginSetupInput)
    {
        throw new ArgumentNullCmfException("BeginSetupInput");
    }
    return true;
```

```
    // Cmf
    UseReference("Cmf.Foundation.Common.dll", "Cmf.Foundation.Common.LocalizationService");
    UseReference("Cmf.Navigo.BusinessOrchestration.dll",
"Cmf.Navigo.BusinessOrchestration.ResourceManagement.InputObjects");
    UseReference("Cmf.Common.CustomActionUtilities.dll", "Cmf.Common.CustomActionUtilities");
    UseReference("Cmf.Common.CustomActionUtilities.dll",
"Cmf.Common.CustomActionUtilities.Abstractions");

    // Other Dependencies
    UseReference("Newtonsoft.Json.dll", "Newtonsoft.Json");

    var serviceProvider = (IServiceProvider)Input["ServiceProvider"];
    var localizationService = serviceProvider.GetService<ILocalizationService>();
    var utilitiesDeeContext = serviceProvider.GetService<IDeeContextUtilities>();

    // Collect inputs
    var resource = (Input["BeginSetupInput"] as BeginSetupInput).Resource as IResource;
```

```
var instance = resource.GetAutomationControllerInstance();
if (instance == null) {
throw new Exception("Resource not connected to any IoT instance");

}
else
{
    string commandMessage = JsonConvert.SerializeObject(
    (
        new Dictionary<string, object>
        {
            { "DEE", "SetSetpoint200" },
            { "Action", "SetSetpoint"},
            { "Value", 200}
        }
    ), Newtonsoft.Json.Formatting.Indented);
    dynamic payload = instance.SendRequest("Cmf.Perform.Action", commandMessage, 10000);

    if(payload == null) {
        throw new Exception("Nothing received" );
    } else if(!bool.Parse(payload["reply"].ToString())) {
        throw new Exception("Setpoint is not 200°C" );
    }
}
```

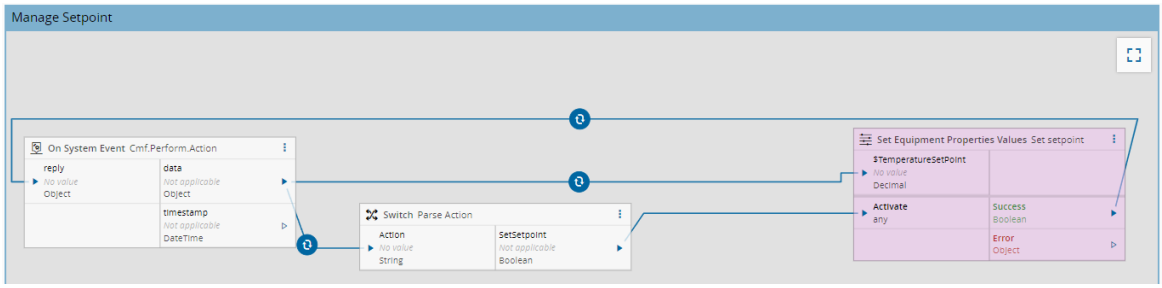
Let's take a look at the code execution. The goal is to send a message to Connect IoT, based on the Resource. In order to perform communication to Connect IoT we will use the message bus through a `Send Request`. The Message Bus supports a "fire and forget" method using `Publish` as well as methods that wait for a success acknowledgement which is the case of `Send Request`.

The execution will send the message to the Controller Instance linked to the resource and will wait for the acknowledge of a reply. It will only succeed if it receives an expected `true` value.

Set Setpoint - Automation Controller

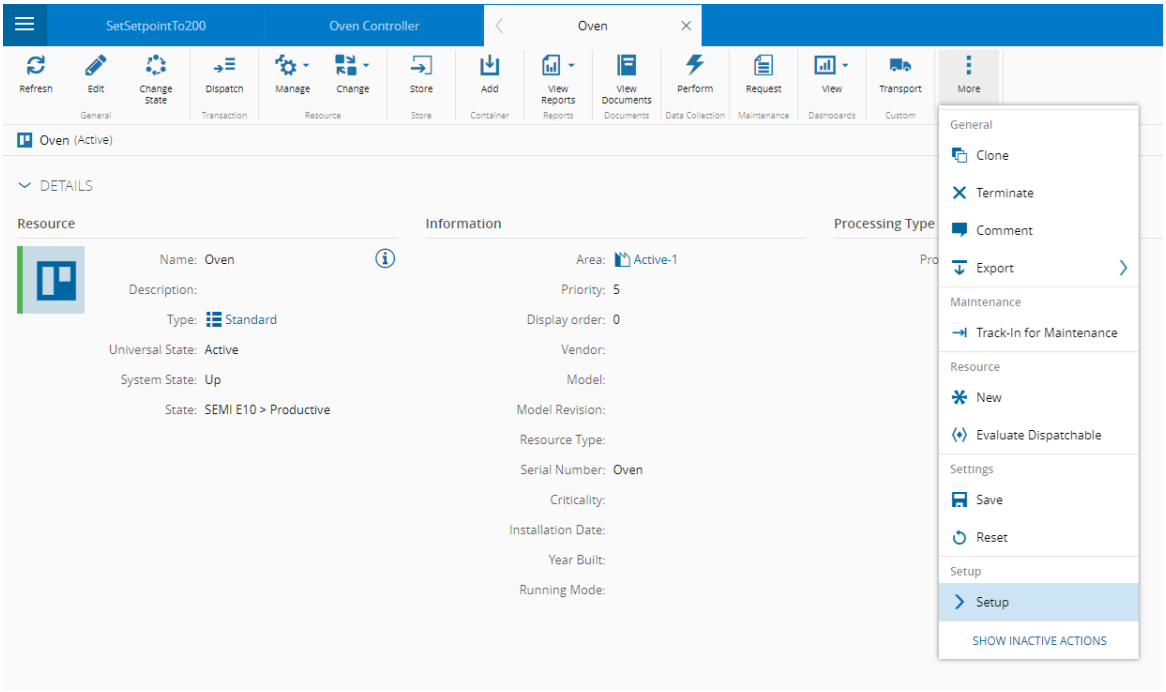
On the Begin Setup we are now broadcasting a message to the Connect IoT layer. Let's now implement the logic of performing equipment integration actions with that information. In IoT we will receive the message and set the setpoint.

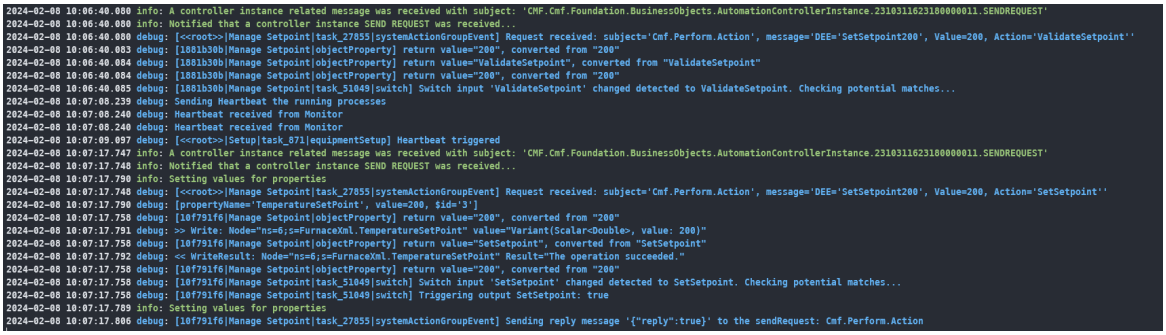
1. Go to `Oven Controller`
2. In the Workflow, create a new page `Manage Setpoint`
3. Drag and drop the following tasks:
   - On System Event: to subscribe to the message bus topic `Cmf.Perform.Action`
   - Set Equipment Properties Values: to set the value of temperature setpoint
4. Go to the `On System Event` settings and for the Action Group, write `Cmf.Perform.Action`.
5. In the `Set Equipment Properties Values`
   a. Add as Input, the Automation Property `TemperatureSetPoint`
6. Link the `On System Event` output `data` to the `Set Equipment Properties Values`
   a. Activate
   b. The `TemperatureSetPoint`
      i. Apply the converter `Get Object Property`
         A. Path `Value`
         B. Type `Decimal`
7. Link the `Set Equipment Properties Values` `Success` to the Reply of the `On System Event` task
   a. Apply the converter `AnyToAny`



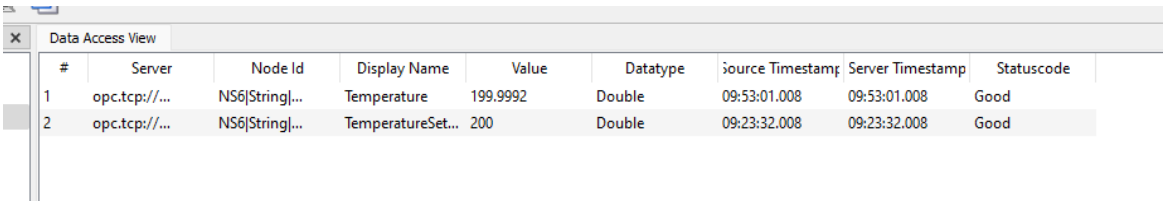
Test with the Automation

In the Resource `Oven`, select `Begin Setup` and press `Begin`.





Notice that in UA Expert, the tag for the temperature setpoint is now 200ºC as expected.



Validate Setpoint - On Resource Complete Setup

In the **Begin Setup**, we've set the setpoint, but we only want the **Complete Setup** to be possible if the temperature matches the setup. We will then use the same DEE to validate the temperature on the Complete Setup.

Change a DEE - SetSetpointTo200

Let's add the Action Group to our DEE.

1. Go to the `Details` tab
2. Press `Add` on the Action Group
3. Search for `ResourceManagement.ResourceManagementOrchestration.CompleteSetup.Post`
4. Check it and press `Add`

If in step 3. the action group is not present:

1. Go to `Administration > DEE Actions`
2. Select the Settings (three vertical dots) next to the `Action Groups`
3. Select `Add new Action Group`
4. Give as `Name` - `ResourceManagement.ResourceManagementOrchestration.CompleteSetup.Post`
5. Select `Create`

Perform the steps described in add the Action Group to the DEE.

```
    /// <summary>
    /// Summary text: Send Setpoint to IoT on Begin Setup
    /// Actions groups:
    ///     * ResourceManagement.ResourceManagementOrchestration.BeginSetup.Post
    /// Depends On:
    /// Is Dependency For:
    /// Exceptions:
    /// </summary>

    var serviceProvider = (IServiceProvider)Input["ServiceProvider"];

    // Validate input
    var hasBeginSetup = Input.ContainsKey("BeginSetupInput") && Input["BeginSetupInput"] is
BeginSetupInput;
    var hasCompleteSetup = Input.ContainsKey("CompleteSetupInput")  && Input["CompleteSetupInput"]
is CompleteSetupInput;

    if (!hasBeginSetup && !hasCompleteSetup)
    {
        throw new ArgumentNullCmfException("SetupInput");
    }

    return true;
```

```
    // Cmf
    UseReference("Cmf.Foundation.Common.dll", "Cmf.Foundation.Common.LocalizationService");
    UseReference("Cmf.Navigo.BusinessOrchestration.dll",
"Cmf.Navigo.BusinessOrchestration.ResourceManagement.InputObjects");

    // Other Dependencies
    UseReference("Newtonsoft.Json.dll", "Newtonsoft.Json");

    var serviceProvider = (IServiceProvider)Input["ServiceProvider"];
    var localizationService = serviceProvider.GetService<ILocalizationService>();

    // Collect inputs
    var hasBeginSetup = Input.ContainsKey("BeginSetupInput") && Input["BeginSetupInput"] is
```

```
BeginSetupInput;
    var resource = hasBeginSetup ? (Input["BeginSetupInput"] as BeginSetupInput).Resource:
(Input["CompleteSetupInput"] as CompleteSetupInput).Resource;
    var action = hasBeginSetup ? "BeginSetup": "CompleteSetup";

    var message =  new Dictionary<string, object>
    {
        { "DEE", "SetSetpoint200" },
        { "Value", 200}
    };

    switch(action)
    {
    case "BeginSetup":
        message.Add("Action", "SetSetpoint");
        break;
    case "CompleteSetup":
        message.Add("Action", "ValidateSetpoint");
        break;
    default:
        throw new Exception("Unknow Action");
    }

    var instance = resource.GetAutomationControllerInstance();
    if (instance == null) {
        throw new Exception("Resource not connected to any IoT instance");
    }
    else
    {
        string commandMessage = JsonConvert.SerializeObject(
        (
           message
        ), Newtonsoft.Json.Formatting.Indented);
        dynamic payload = instance.SendRequest("Cmf.Perform.Action", commandMessage, 10000);

        if(payload == null) {
            throw new Exception("Nothing received" );
        } else if(!bool.Parse(payload["reply"].ToString())) {
            throw new Exception("Setpoint is not 200°C"+payload["reply"].ToString() );
        }
    }
```

Notice the big change in the execution. We kept most of our code exactly the same but added a new context: the action. This action will be used to provide context to the Connect IoT layer to know that in one situation it must set the setpoint and in another situation it will validate that the temperature is in the setpoint.

Validate Setpoint - Automation Controller

On the complete setup we are now broadcasting a message to the Connect IoT layer. Let's now validate the setpoint against the temperature.

1. Go to `Oven Controller`
2. In the Workflow, got to page `Manage Setpoint`
3. Drag and drop the following tasks:
   - Switch: tto allow us to perform conditional Actions
   - Get Equipment Properties Values: to set the value of temperature setpoint
   - Expression Evaluator: to compare te setpoint against the temperature
4. In the `Switch` task add
   a. Input `Action` of Type `String`
   b. Add Outputs
      i. SetSetpoint

A. Equals - SetSetpoint

B. Name - SetSetpoint

C. Type - Boolean

D. Value - `true`

ii. ValidateSetpoint

A. Equals - ValidateSetpoint

B. Name - ValidateSetpoint

C. Type - Boolean

D. Value - `true`

5. Link the output `data` to the `Switch` input Action
   a. Apply converter `GetObjectProperty`
      i. Path - `Action`
      ii. Type - `String`
6. Link the output `SetSetpoint` to the `Activate` of the `Set Equipment Properties Values` task
7. Link the output `ValidateSetpoint` to the `Activate` of the `Get Equipment Properties Values` task
8. In the `Expression Evaluator`
   a. Check the flag Clear inputs to `false`
   b. Add Inputs
      i. Setpoint
         A. Name - `Setpoint`
         B. Type - `Decimal`
         C. Default Value - 0
      ii. Temperature
         A. Name - `Temperature`
         B. Type - `Decimal`
         C. Default Value - 0
   c. Add Outputs
      i. IsValid
         A. Name - `IsValid`
         B. Type - `String`
         C. Expression:

```
abs(Setpoint - Temperature) < 1
```

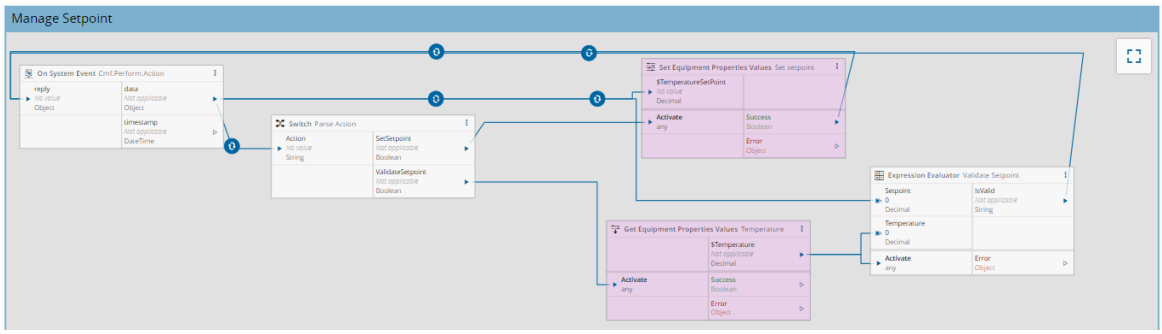
Notice that we are not doing an exact match, this is because the temperature oscillates between the setpoint. So we are putting a threshold of accepted values.

9. Link the `$Temperature` output
   a. To the input Temperature of the `Validate Setpoint` task
   b. To the input Activate of the `Validate Setpoint` task
10. Link the output `data` to the `Expression Evaluator` input `Setpoint`
    a. Apply converter `GetObjectProperty`

i. Path - `Value`

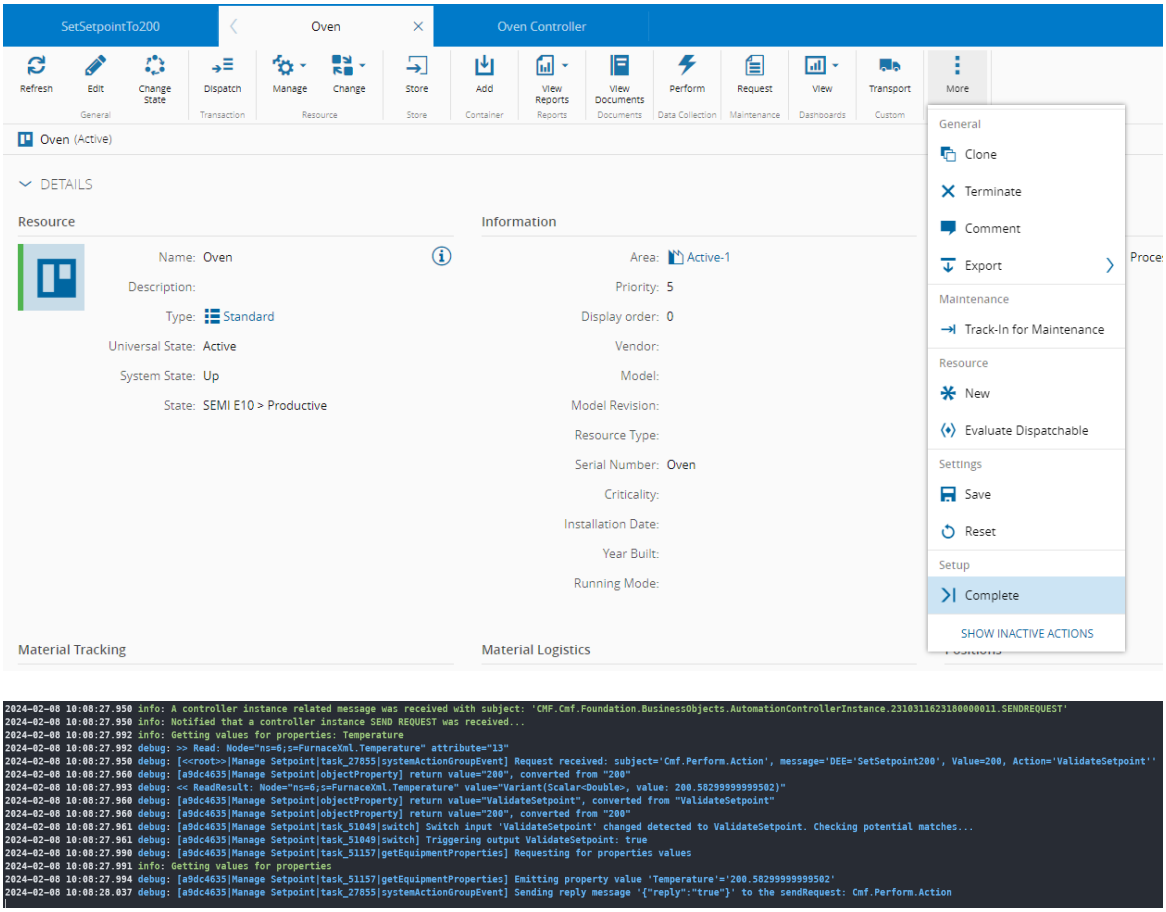
ii. Type - `Decimal`

11. Link the `Get Equipment Properties Values` to the
    a. Input `$Temperature` of the `Expression Evaluator`
    b. Input `Activate` of the `Expression Evaluator`
12. Link the `Expression Evaluator` output `IsValid` to the input `reply` of the `On System Event`



Test with the Automation

In the Resource `Oven`, if you have already set performed the Begin Setup operation on the Resource, perform the Complete Setup. If the temperature is now 1+-200ºC you should be successful, if not it will show an error message.




```
2024-02-08 10:08:27.950 info: A controller instance related message was received with subject: 'CMF.Cmf.Foundation.BusinessObjects.AutomationControllerInstance.231031162318000011.SENDREQUEST'
2024-02-08 10:08:27.950 info: Notified that a controller instance SEND REQUEST was received...
2024-02-08 10:08:27.992 info: Getting values for properties: Temperature
2024-02-08 10:08:27.992 debug: >> Read: Node="ns=6;s=FurnaceXml.Temperature" attribute="13"
2024-02-08 10:08:27.950 debug: [<<root>>|Manage Setpoint|task_27855|systemActionGroupEvent] Request received: subject='Cmf.Perform.Action', message='DEE='SetSetpoint200', Value=200, Action='ValidateSetpoint''
2024-02-08 10:08:27.960 debug: [a9dc4635|Manage Setpoint|objectProperty] return value="200", converted from "200"
2024-02-08 10:08:27.993 debug: << ReadResult: Node="ns=6;s=FurnaceXml.Temperature" value="Variant(Scalar<Double>, value: 200.50299999999502)"
2024-02-08 10:08:27.960 debug: [a9dc4635|Manage Setpoint|objectProperty] return value="ValidateSetpoint", converted from "ValidateSetpoint"
2024-02-08 10:08:27.960 debug: [a9dc4635|Manage Setpoint|objectProperty] return value="200", converted from "200"
2024-02-08 10:08:27.961 debug: [a9dc4635|Manage Setpoint|task_51049|switch] Switch input 'ValidateSetpoint' changed detected to ValidateSetpoint. Checking potential matches...
2024-02-08 10:08:27.961 debug: [a9dc4635|Manage Setpoint|task_51049|switch] Triggering output ValidateSetpoint: true
2024-02-08 10:08:27.990 debug: [a9dc4635|Manage Setpoint|task_51157|getEquipmentProperties] Requesting for properties values
2024-02-08 10:08:27.991 info: Getting values for properties
2024-02-08 10:08:27.994 debug: [a9dc4635|Manage Setpoint|task_51157|getEquipmentProperties] Emitting property value 'Temperature'='200.50299999999502'
2024-02-08 10:08:28.037 debug: [a9dc4635|Manage Setpoint|task_27855|systemActionGroupEvent] Sending reply message '{"reply":"true"}' to the sendRequest: Cmf.Perform.Action
```

Post Data - On TrackIn

As we saw when we were collecting values, the temperature varies in a threshold of the setpoint. So we want to collect the temperature at the time of the TrackIn, to keep track of the temperature that the material will be exposed at the time of the TrackIn.

One of the important things to consider in a DEE is that it will run every time the action is performed in the system. In the case that we are addressing we wish to be more specific and require that the action be performed only in particular steps. In order to achieve this, let´s add an attribute in our Step to flag it as a Step where we want the logic to execute.

Create a Step Attribute

1. Go to `Administration > Entity Types`
2. Select the `Step`
3. In the Attributes, select `Manage`
4. Press the plus button to add
5. Create a new attribute
    a. Name `NotifyEQOnTrackIn`
    b. Scalar Type `Bit`
    c. Update
6. Press the `Generate` button
7. Go to the `Oven Step`
8. The attribute `NotifyEQOnTrackIn` will now be visible under the list of Attributes
9. `Edit` the attribute and change it to be `true`

Create a DEE - NotifyIoTOnEquipmentTrackIn

1. Go to `Administration > DEE Actions`
2. Select `New`
3. Give as Name `NotifyIoTOnEquipmentTrackIn`, for now the classification and action group is not important

Let's add the Action Group to our DEE.

1. Go to the `Details` tab
2. Press `Add` on the Action Group
3. Search for `BusinessObjects.MaterialCollection.TrackIn.Pre`
4. Check it and press `Add`

If in step 3. the action group is not present:

1. Go to `Administration > DEE Actions`
2. Select the Settings (three vertical dots) next to the `Action Groups`
3. Select `Add new Action Group`
4. Give as `Name` - `BusinessObjects.MaterialCollection.TrackIn.Pre`
5. Select `Create`

Starting on the code for the `Test Condition Code`. We want to validate that the Inputs that we are interested are correct, then check the Step of the Material to validate whether we should process it and subsequently pass that value to our DEE context.

```
    /// <summary>
    /// Summary text: Start production on material track in
```

```
    /// Actions groups:
    ///     * BusinessObjects.MaterialCollection.TrackIn.Pre
    /// Depends On:
    /// Is Dependency For:
    /// Exceptions:
    /// </summary>

    // Retrieve from Dependency Injection Container
    var serviceProvider = (IServiceProvider)Input["ServiceProvider"];

    bool isToExecute = false;

    // Validate Inputs
    if (Input["MaterialCollection"] is not IMaterialCollection materialCollection){
        throw new ArgumentNullCmfException("MaterialCollection");
    }

    if (Input["Resource"] is not IResource resource) {
        throw new ArgumentNullCmfException("Resource");
    }

    resource.Load();

    var step = materialCollection.FirstOrDefault().Step;
    step.LoadAttributes(new Collection<string>() {"NotifyEQOnTrackIn"});

    // Only notify IoT if it's a notifyIoTStep
    if (step.Attributes != null && step.Attributes.ContainsKey("NotifyEQOnTrackIn") &&
(bool)step.Attributes["NotifyEQOnTrackIn"])
    {
        isToExecute = true;
    }

    return isToExecute;
```

In the execution we will iterate through the **Materials** and for each one we will send a message to the Controller Instance linked to the **Resource** of the **Material** and wait for success. Bear in mind that if the wait time is very long or if no response is sent back from Connect IoT, the GUI will be in a loading screen and then eventually timeout, consider always replying back and handling the exception.

> **Note**
>
> By using the `resource.GetAutomationControllerInstance();` and using then the instance to do a `instance.SendRequest`, you guarantee that only the controller instance associated to this Resource will be notified. If you want to broadcast every listener consider using the MessageBus native methods.

```
    // Other Dependencies
    UseReference("Newtonsoft.Json.dll", "Newtonsoft.Json");

    // Retrieve from Dependency Injection Container
    var serviceProvider = (IServiceProvider)Input["ServiceProvider"];

    // Collect inputs
    var resource = Input["Resource"] as IResource;
    var materialsToStartProduction = Input["MaterialCollection"] as IMaterialCollection;

    // Iterate through materials and send a message to IoT
    foreach (var Material in materialsToStartProduction)
    {
        string commandMessage = JsonConvert.SerializeObject(
            (
                new Dictionary<string, object>
                {
                    { "DEE", "NotifyIoTOnEquipmentTrackIn" },
                    { "Action", "PostToDataCollection"}
                }
            ), Newtonsoft.Json.Formatting.Indented);
```

```
        var instance = resource.GetAutomationControllerInstance();
        if (instance == null) {
            throw new Exception("Resource not connected to any IoT instance");
        }
        else
        {
            dynamic reply = instance.SendRequest("Cmf.Perform.Action", commandMessage, 10000);

            if(reply == null) {
                throw new Exception("Nothing received" );
            }
        }
    }
```

In this example, let's choose a generic topic and a more complex payload. It is also correct to use the topic as the context for the action to be performed like `Cmf.Post.Temperature`. The goal was to demonstrate a more complex payload example.

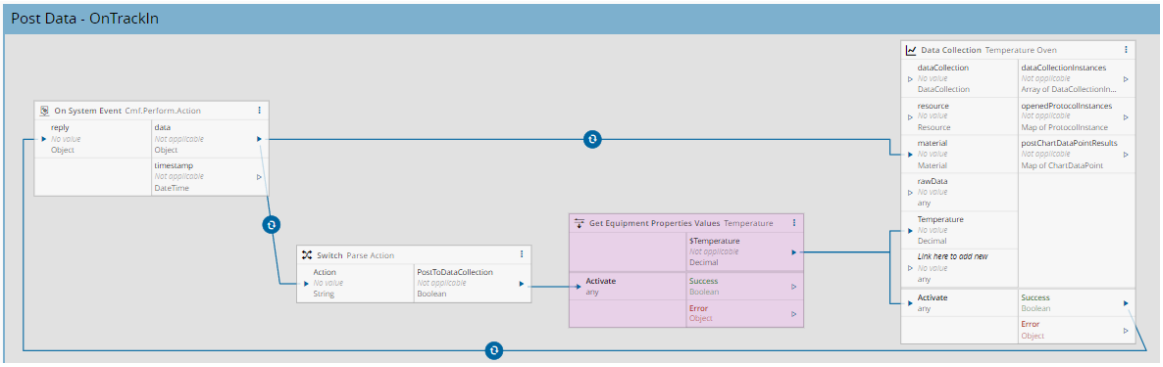
Automation Controller - Implement Post Data

Now that we have a DEE that will notify Connect IoT whenever there is a TrackIn. We will need to create a listener in the Oven Automation Controller and to perform a Post to the DataCollection.

1. Go to `Oven Controller`
2. In the Workflow, create a new page `Post Data - OnTrackIn`
3. Drag and drop the following tasks:
   - On System Event: to subscribe to the message bus topic `Cmf.Perform.Action`
   - Switch: to allow us to perform conditional Actions
   - Get Equipment Properties Values: to retrieve the value of temperature
   - Data Collection: to post data to a Data Collection
4. Go to the `On System Event` settings and for the Action Group, write `Cmf.Perform.Action`.
5. Go to the `Switch` settings
   a. Add Input `Action` as String
   b. Add Outputs
      i. Equals to `PostToDataCollection`
      ii. Name `PostToDataCollection`
      iii. Type `Boolean`
      iv. Value `true`
6. Link the `On System Event` output `data` to the `Switch` input `Action`
   a. Apply the converter `Get Object Property`
      i. Path `Action`
      ii. Type `String
7. Link the `Switch` output `PostToDataCollection` to the `Activate` of the `Get Equipment Properties Values`
8. Go to the `Get Equipment Properties Values` settings and add in the outputs the Automation Property `Temperature`
9. Go to the `Data Collection` and configure like in the Intermediate Connect IoT Tutorial, but change the `Complex Perform Data Collection Mode` to `Perform To Material`
10. Link the `Get Equipment Properties Values` Temperature output
    a. To the `Data Collection` Temperature input

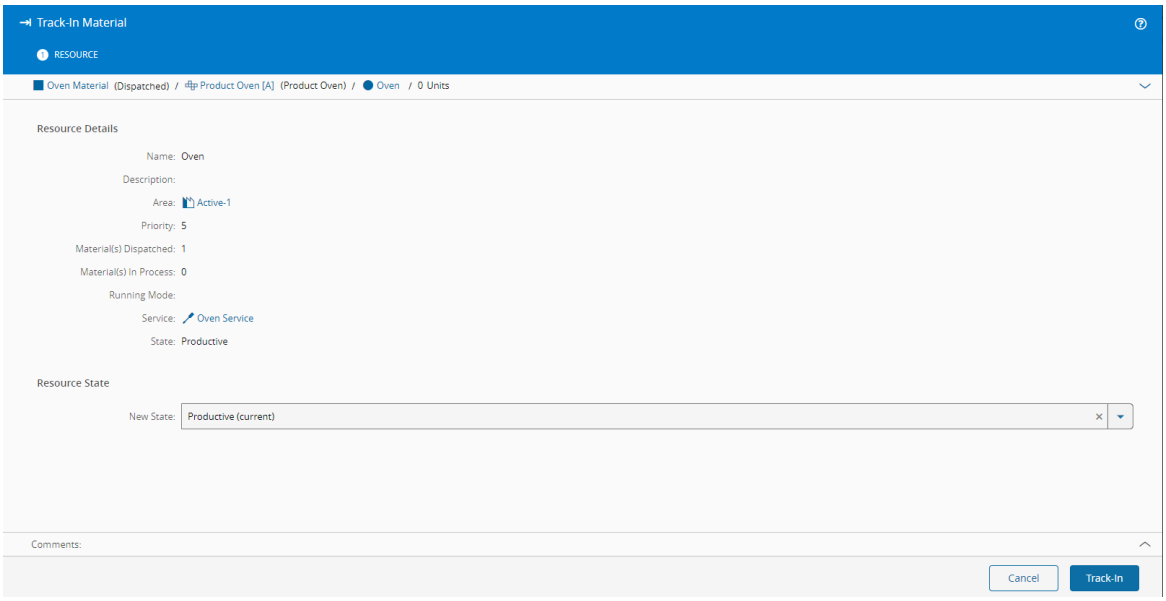
b. To the `Data Collection` active input

11. Link the `Success` output of the `Data Collection` task to the Reply of the `On System Event` task
12. Link the `data` output of the `OnSystemEvent` task to the `material` of the `Data Collection` task
    a. Apply the converter `Get Object Property`
        i. Path `Material`
        ii. Type `String`
    b. Apply the converter `CreateSystemEntity`
        i. entityType `Material`
        ii. identifier: `Name`

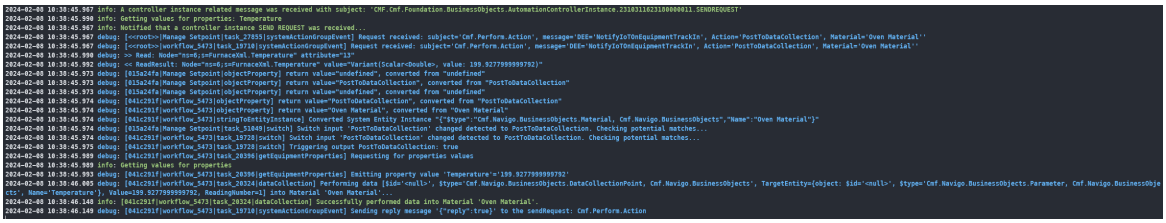


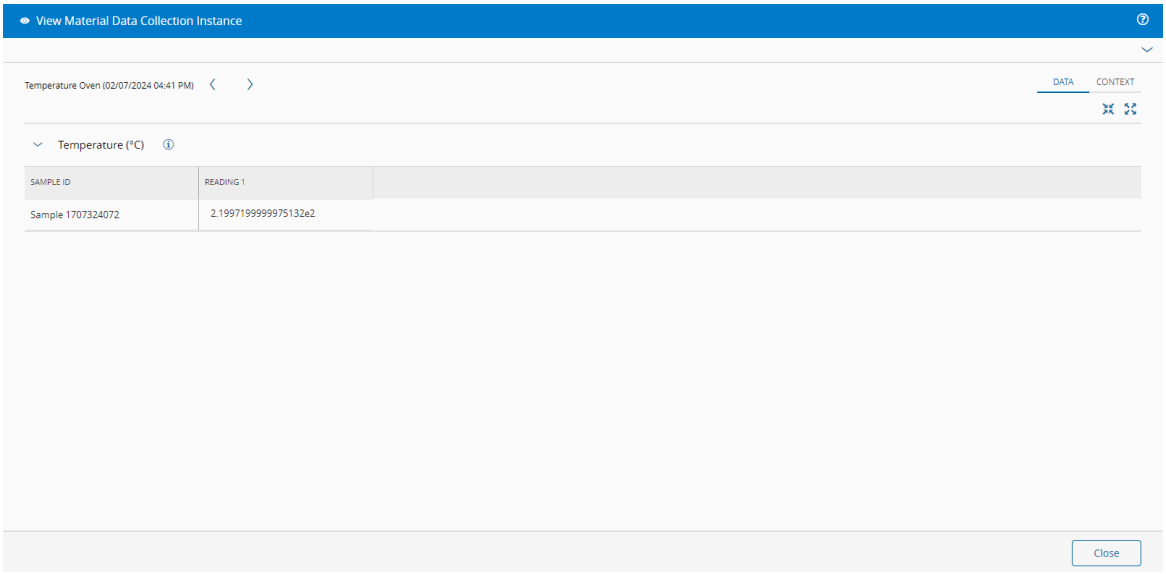
Create a Material and Dispatch and TrackIn

Create a Material `Oven Material` with the Product `Product Oven` and dispatch it for the Resource `Oven`. Now track in the Material at the Resource `Oven`.



Now, if we go to the collected data of the **Material** it will show the data collection `Temperature Oven` with the value `Temperate`. Currently, the OPC-UA server is sending a temperature value with a large resolution, however if you prefer a more readable value feel free to apply an `Expression Evaluator` task to round up the value.





Automatic TrackOut - After 1 minute

The material was tracked in successfully and we can now say that is being processed. We want to have a limited amount of time during which the material can be exposed to the setpoint temperature and then perform a TrackOut directly from the automation. We can use the default services provided out-of-the-box by the system and custom services, but for this case let's do a similar approach to the GUI. Using a DEE also allows us the opportunity to do some further customization if we wish to.

Create a DEE - PerformATrackOut

Let's create a DEE to retrieve the Temperature from Connect IoT.

1. Go to `Administration > DEE Actions`
2. Select `New`
3. Give as Name `PerformATrackOut`
4. Give as Classification `ConnectIoT`

```
// Retrieve Service Provider
var serviceProvider = (IServiceProvider)Input["ServiceProvider"];

IMaterial material = serviceProvider.GetService<IMaterial>();

// Retrieve Material Name from the Inputs
material.Name = Input["Material"] as string;
material.Load();

material.TrackOut();
```

In the DEE execution, which will receive a Material name, load the Material object and execute the TrackOut.

1. Go to `BusinessData > Rule`
2. Select `New`
3. Give as Name `PerformATrackOut`
4. Give as Scope `ConnectIoT`
5. DEE Action `PerformATrackOut` - if it does not appear in the search box, validate that the DEE Classification is correct

Automation Controller - TrackOut Automatically After 1 min

In this implementation we will use the same topic as the TrackIn, we will start a timer that will trigger the Rule `PerformATrackOut` after 1 minute. We will require the Material name, but only after a minute, to perform the TrackOut.

Connect IoT executes the tasks asynchronously, depending on activation time. In cases where we know before hand that there is a big time frame difference and that inputs can be overriden while waiting for the termination of other tasks, it's strongly recommended to use either the `Synchronize` task or sub-workflows. The sub-workflows preserve the activation context when activated, so there is no problem to receive the Material and then wait for 1 minute, as the context for that sub-workflow will be preserved and further activations will be new instances of the sub-workflow. The sub-workflow has an execution timeout, so the execution may not exceed that timeout.

1. Go to `Oven Controller`
2. In the Workflow, create a new page `TrackOut Automatically - After 1 min`
3. Drag and drop the following tasks:
   - On System Event: to subscribe to the message bus topic `Cmf.Perform.Action`
   - Switch: to allow us to perform conditional Actions
   - Workflow: to execute sub-workflows
4. Go to the `On System Event` settings and for the Action Group, write `Cmf.Perform.Action`.
5. Go to the `Switch` settings
   a. Add Input `Action` as String
   b. Add Outputs
      i. Equals to `PostToDataCollection`
      ii. Name `PostToDataCollection`
      iii. Type `Boolean`
      iv. Value `true`
6. Link the `On System Event` output `data` to the `Switch` input `Action`
   a. Apply the converter `Get Object Property`
      i. Path `Action`
      ii. Type `String
7. In the Workflow, create a new page `Sub - TrackOut Automatically - After 1 min`
8. Drag and drop the following tasks:
   - `Start`: the start context of the sub-workflow
   - `End`: the end result of the sub-workflow
   - `Timer`: to allow us to perform conditional Actions
   - `Execute Action`: to execute sub-workflows
9. In the `Timer` task open the `Settings`

a. Set the `Auto activate` to `false`

b. Set the `interval` to `60000`

c. Set the `Working Mode` to `Number of Occurrences` to `1`

10. Link the output `Success` of the `Timer` to the `Activate` of the `Execute Action`
11. Open the `Execute Action` settings

a. Select the `Rule` - `PerformATrackOut`

b. In the `Inputs` tab

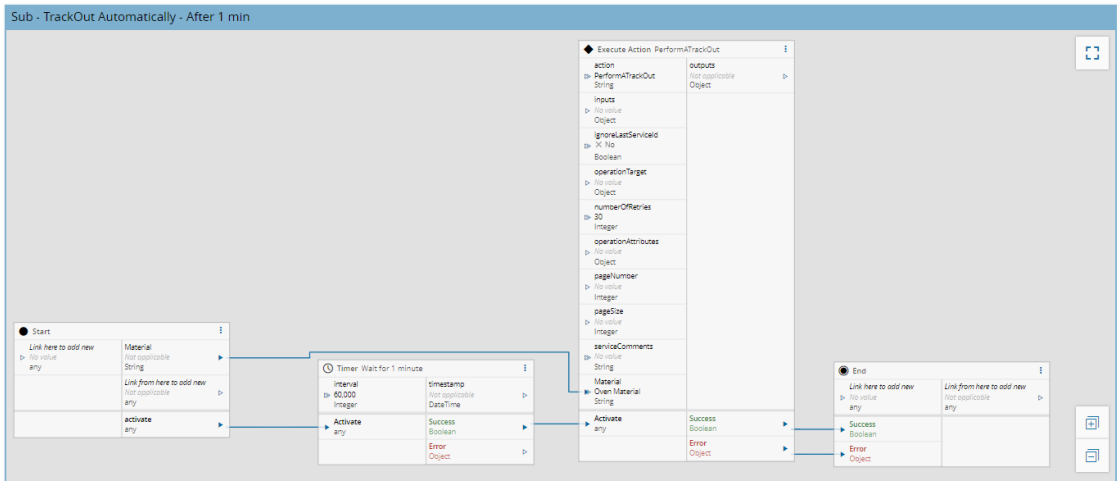
i. Add a new input

A. Name - `Material`

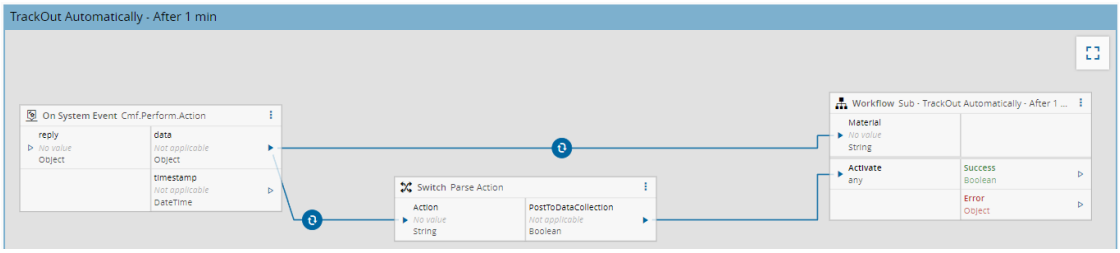
B. Type - `String`

C. Default Value - `N/A` - can be whatever you wish as it wil be overriden

12. Link the output `Material` of the task `Start` to the `Material` Input of the `Execute Action`
13. Link the output `Success` and `Error` to the matching inputs of the `End` task
14. Go to the page `TrackOut Automatically - After 1 min`

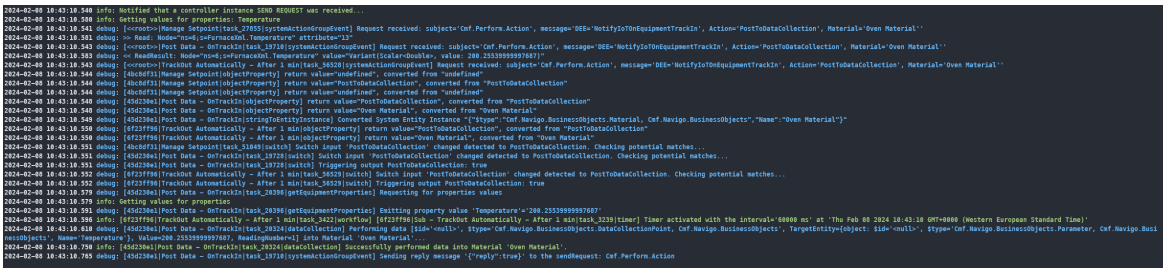


15. In the `Workflow` task settings, select the `Automation Workflow` `Sub - TrackOut Automatically - After 1 min`
16. Link the output `data` of the `On System Event` to the input `Material` of the `Workflow` task

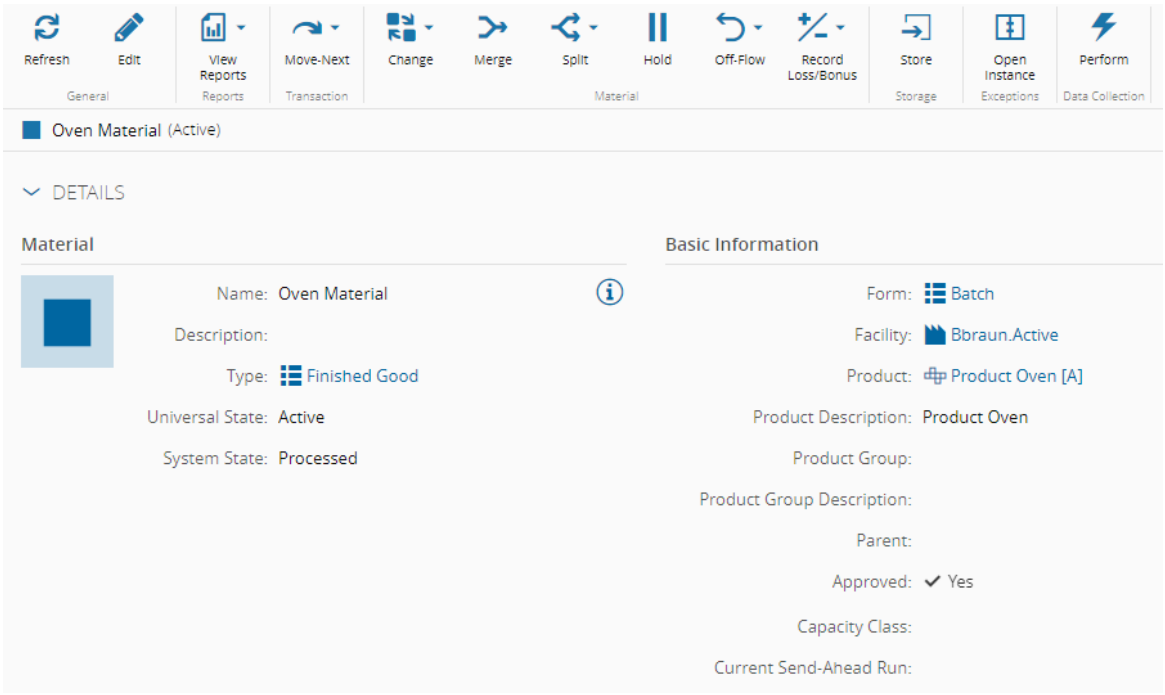


Interestingly, look that we are not replying back, even though the message was a `SendRequest`, it's not a problem, because the `SendRequest` works as a success on the first reply, and that will be achieved by the workflow `Post Data - OnTrackIn`.

In order to test, let´s `Abort` our Material, `Dispatch` and `TrackIn` and let´s see the differences.



We can still see the previous reply, but now we register a new handler for our TrackOut. Then, after 1 minute:



You now have a built structure using Connect IoT to track the lifecycle of a Material lifecycle, the lifecycle of a Resource and to collect values. This is the end of the advanced 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.

Copyright © 2023 Critical Manufacturing. All rights reserved.