Critical manufacturing
an ASM PT company

Talking to MES

12.0.1

November 2026

DOCUMENT ACCESS

Public

DISCLAIMER

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

Connect IoT - Talking to MES Tutorial

Overview

A controller and MES talk through three channels, each with its own tasks:

Channel	Who starts it	Tasks
The workflow asks MES	The workflow	`Execute Service Call`, `Execute Action`, `Entity Instance`, `Get Configurations`, `Create Notification`, `Adjust State`, `API Post Event`
MES asks this controller instance	MES, for example a DEE action that calls `SendRequest` or `Publish` on the instance	`System Action Listener` with `Send System Action Reply`, and `Ready State Requested` with `Send Ready State Reply`
The message bus	Any process that sends on a subject	`Send Message Bus Message`, and `Message Bus Listener` with `Message Bus Reply`

A system action reaches only the instance it is sent to, and its action group only tells the messages of that instance apart. A message bus subject is global: every listener of the subject hears it. That difference decides the channel when MES has something to tell the line.

In this tutorial, you add one Data Flow page to an existing **Automation Controller**. A DEE action asks the **Automation Controller Instance** three questions, and the workflow answers each one: a load limit read from a configuration entry, the **Resource** that the instance represents, and a recipe obtained over the message bus. The workflow also raises a notification for each limit request. The tutorial is for integrators who completed the basic configuration tutorial. It extends Connect IoT -

MES Interoperability Scenarios Tutorial, where MES transactions send the requests, with the general request and reply tasks.

> **Info**
>
> For more information, see System Action Listener, Message Bus Listener, and Retry Mechanism.

Before You Start

- Have an **Automation Controller** with the **Workflow Type** `Data Flow`, for example from Connect IoT - Basic Configuration Tutorial.
- Have an **Automation Controller Instance** of that controller connected to a **Resource**, such as `Resource-A`.
- Have an **Automation Manager** that you can start in console mode.
- Make sure you can create and execute DEE actions (DEE Actions) and create configuration entries (Configuration).
- Check the values of the `NotificationType` and `NotificationSeverity` lookup tables of your installation, such as `General` and `Information`.

Sequence of Steps

Step 1: Create a DEE Action That Asks the Instance

A DEE action reaches an **Automation Controller Instance** through the **Resource** it is connected to. `SendRequest` sends a request under an action group and waits for the answer; `Publish` sends a notification that nothing answers.

Go to **Administration** > **DEE Actions**, select **New**, and create the action `AskController` with the classification `ConnectIoT` and this code:

```
UseReference("Cmf.Foundation.BusinessObjects.dll",
"Cmf.Foundation.BusinessObjects");
UseReference("Cmf.Foundation.BusinessOrchestration.dll", "");
UseReference("", "Cmf.Foundation.Common.Exceptions");
UseReference("", "Cmf.Foundation.Common");
UseReference("Cmf.Navigo.BusinessObjects.dll", "Cmf.Navigo.BusinessObjects");
UseReference("%MicrosoftNetPath%Microsoft.CSharp.dll", "");

var serviceProvider = (IServiceProvider)Input["ServiceProvider"];
IResource resource = serviceProvider.GetService<IResource>();
resource.Name = Input["Resource"] as string;
resource.Load();
```

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

// Ask this instance only, under the action group given, and wait up to 10
seconds
dynamic reply = instance.SendRequest(Input["Group"] as string, null, 10000);
Input.Add("Answer", reply == null ? "Nothing received" : reply.ToString());
```

Select **Validate**. The syntax check passes, and the action is ready to execute with the inputs `Resource` and `Group`.

Step 2: Answer a Request from MES

Open the controller, add a Data Flow page named `Talk to MES`, and place the tasks below:

Task	Settings to set	Inputs and outputs to link
`System Action Listener`, named `On limit request`	**Action Group** `load.limit`; keep **Auto Enable** on and **Reply Timeout** `60000`	-
`Send System Action Reply`, named `Answer the limit`	Keep **Default Reply** `{}`	`Success` of `On limit request` to `Activate`

Select **Save**, and stop and start the **Automation Controller Instance** in **Automation** > **Connect IoT** so that it runs the new page. Repeat this after each later change to the workflow.

Execute `AskController` with `Resource` set to `Resource-A` and `Group` set to `load.limit`. The **Answer** output holds `{}`: the reply received no value, so the task sent its **Default Reply**.

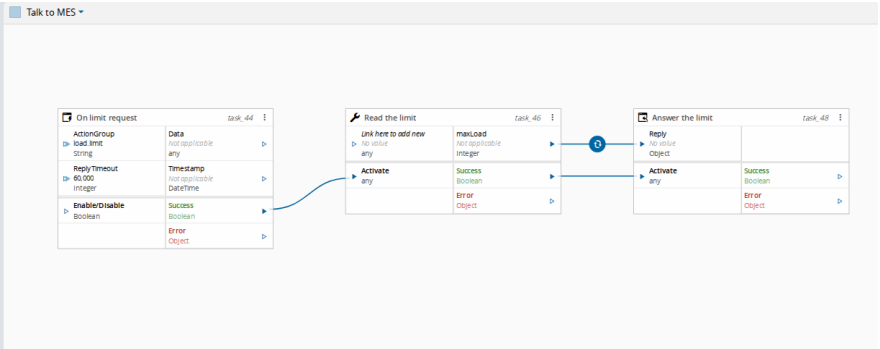
> **Warning**
>
> A reply task answers the request of the run in which it is activated. Activate it from the run that the listener started, as here; activated anywhere else, it emits an error and MES receives nothing. See Zones.

Step 3: Answer with a Configuration Value

`Get Configurations` reads configuration entries, values that belong to the installation and change without a new controller version.

1. Go to **Administration** > **Configuration**, create the entries for the path `/Cmf/Custom/ConnectIoT/`, and create in it the entry `MaxLoad` with the type `String` and the value `25`.
2. On the `Talk to MES` page, add `Get Configurations`, named `Read the limit`, and on its **Outputs** tab add the output `maxLoad` with the type `Integer`, the full path `/Cmf/Custom/ConnectIoT/MaxLoad`, and the default value `-1`. The canvas shows the output as `maxLoad`, and its stored port name is `maxLoadOut`.
3. Replace the link into `Activate` of `Answer the limit`, and link the tasks as follows:

From	To
`Success` of `On limit request`	`Activate` of `Read the limit`
`maxLoad` of `Read the limit`	`Reply` of `Answer the limit`, with the `Any To Any` converter
`Success` of `Read the limit`	`Activate` of `Answer the limit`



Execute `AskController` again with the group `load.limit`. The answer holds `reply` with the value `25`: an object is sent as it is, and any other value is wrapped as `{"reply": <value>}`. The value is converted to the type of the output, and an entry that is missing or empty emits the default value with a warning, followed by `Success`.

Edit `MaxLoad` to `0`, and execute the action once more without restarting the instance. The answer holds `reply` with the value `0`, read on this request. In 12.0, `Send System Action Reply`, `Message Bus Reply`, and `Send Ready State Reply` send the **Default Reply** only when no value arrived since their previous answer, so `0` and `false` are sent as answers.

Step 4: Tell the Operators with a Notification

Add `Create Notification`, named `Notify the request`, and link `Success` of `Read the limit` to its `Activate`. In its settings:

- Leave **Name** empty, so that MES names each notification. MES holds one notification per name, so a fixed name creates the first notification and fails every later one.
- Set **Type** to `General`, **Title** to `MES asked for the load limit`, and **Severity** to a value of your installation, such as `Information`.
- Set **Assignment** to `Everyone`, and keep the clearance **Mode** `ManualSingleUser`, which needs no **Valid To** date.

Execute `AskController` twice with the group `load.limit`. Each execution adds a new notification for every user.

Step 5: Hand Over the Resource of the Instance

`Entity Instance` hands the workflow the entity that the **Automation Controller Instance**, or one of its driver instances, is attached to, so one workflow serves every instance.

1. Add a second `System Action Listener`, named `On resource request`, with the **Action Group** `resource.describe`.
2. Add `Entity Instance`, named `This instance's Resource`, and choose the entity of the controller when the designer asks.
3. Add a second `Send System Action Reply`, named `Answer with the Resource`.
4. Link `Success` of the listener to `Activate` of `Entity Instance`, `instance` to `Reply`, and `Success` of `Entity Instance` to `Activate` of `Answer with the Resource`.

Execute `AskController` with the group `resource.describe`. The answer holds the **Resource**, with the `Name` `Resource-A`.

Choose the task that calls MES from what it needs to reach:

Task	Use it when	What to know
`Entity Instance`	You need the entity of this instance or of a driver instance	Loaded once and kept: later activations call MES only with **Reload in every change** on
`Execute Service Call`	One MES service answers the need, such as `GetObjectByName` for an entity that a request names	Each input port carries the bare property name of the service, such as `Name`. Each output port is named `output$<property>`, such as `output$Instance`, read in Control Flow as `{{ $<block name>.output$Instance }}`
`Execute Action`	The work needs several services, or one transaction	Runs the DEE action of a **Rule** with the scope `ConnectIoT`, with one added input per value. The `inputs` input accepts only a `Map`

In Data Flow, an added input of `Execute Service Call` or `Execute Action` keeps the last value it received, so deliver a value on every activation.

Step 6: Ask over the Message Bus

A message bus request is answered by whichever listener subscribes to its subject: another controller, an MES application, or, here, the same workflow.

Task	Settings to set	Inputs and outputs to link
`Message Bus Listener`, named `Recipe service`	**Subject** `recipe.lookup`	-
`Message Bus Reply`, named `Answer recipe questions`	**Default Reply** `{"recipe": "R-7"}`	`Success` of `Recipe service` to `Activate`
`System Action Listener`, named `On recipe request`	**Action Group** `recipe.describe`	-

Task	Settings to set	Inputs and outputs to link
`Send Message Bus Message`, named `Ask for the recipe`	**Message Type** `Request`, **Subject** `recipe.lookup`, **Message** `{"product": "P-100"}`, **Timeout** `5000`	`Success` of `On recipe request` to `Activate`
`Send System Action Reply`, named `Answer with the recipe`	-	`Reply` of `Ask for the recipe` to `Reply`; `Success` of `Ask for the recipe` to `Activate`

Execute `AskController` with the group `recipe.describe`. The answer holds `recipe` with the value `R-7`. A `Notification` message type is sent without waiting, and its `Success` means the message was handed to the bus, not that anyone received it. A `Request` waits up to **Timeout** for an answer and emits `Error` when none arrives.

> **Note**
>
> Send an MES identifier inside an object, such as `{"Id": "2102050000000000003"}`. Text received on the `Message` input is parsed as JSON, so a bare 19-digit identifier becomes a number that loses its last digits, and text that is not JSON is sent as `{}`.

Validate the Result

Execute `AskController` with `Resource` set to `Resource-A` and each group:

Group	Answer	Also
`load.limit`	`reply` with the current value of `MaxLoad`	A new notification
`resource.describe`	The **Resource** `Resource-A`	-
`recipe.describe`	`recipe` with the value `R-7`	-
`nothing.here`	`Nothing received`, after the 10-	No listener subscribes to

Group	Answer	Also
	second timeout	the group

When nothing answers within the timeout, `SendRequest` returns `null`, and the action reports `Nothing received`. It raises an exception only when the workflow replies with an error.

When a Call to MES Fails

The tasks that call MES share three retry settings:

- **Number of Retries** (default `30`) counts the attempts of the controller, made only for the transient failures listed in Retry Mechanism, such as a deadlock, data changed since last viewed, or a refused connection. Any other failure is reported once on `Error`.
- **Sleep Time between retries (ms)** (default `1000`) is the wait after each failed attempt.
- **Number of System Retries** (default `10`) is meant for MES, which retries the service on its side. In 12.0, `Execute Action`, `API Post Event`, `Get Configurations`, and `Resolve Table` send it to MES. On `Execute Action` and `API Post Event`, a value received on the `numberOfRetries` input replaces the setting.

`Execute Service Call` always sends `10` to MES, whatever its Number of System Retries holds, and `Entity Instance` and `Adjust State` do not send the setting. **Number of Retries** and **Sleep Time between retries (ms)** work on every task that calls MES.

Link `Error` wherever a failure matters. In Control Flow, an error ends the branch. `Execute Service Call` with no service selected is an exception: it emits a text on `Error`, and a Control Flow branch continues as if the call succeeded.

Control Flow Differences

Topic	Data Flow	Control Flow
Answering a request	`Send System Action` `Reply` or `Message Bus` `Reply` in the run of the listener	The value returned by the `handler` branch: an object as it is, any other value as `{"reply": <value>}`, and

Topic	Data Flow	Control Flow
		`{}` when the branch ends without `Return`
Answering a ready-state request	`Send Ready State Reply`	The returned value; `false` when the branch ends without `Return`
Reply Timeout passes	The listener emits `Error` and answers with an error; a later reply is still sent	The branch stops, and the listener answers with an error
Switching a listener on and off	The `Enable/Disable` input	Keep **Auto Enable** on and set the group or subject in the settings

Give each request group or subject one listener that answers: two listeners on one group both answer the same request. Two listeners on a notification are safe, because both hear it and neither answers.

Zero downtime uses the same pattern: when an **Automation Scheduled Action** uses the `DefinedInWorkflow` detector, MES sends a `CheckForReadyState` request, `Ready State Requested` receives it, and the workflow answers with `Send Ready State Reply`. Anything but `true`, including a timeout, counts as not ready. See Scenario 4 - Controller Version Update - Workflow Ready Signal.

Next Steps

Enable the out-of-the-box interoperability scenarios of a **Resource** in Connect IoT - MES Interoperability Scenarios Tutorial, which use the same request and reply pattern for setup, material, container, and recipe transactions. To understand why a Data Flow task sends the last value it held, see Connect IoT - How Values Travel Tutorial, and to build the same answers in Control Flow, see Connect IoT - Control Flow in Depth 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.