Connect IoT - Upload Recipe to Equipment Tutorial

11.3

April 2026

DOCUMENT ACCESS

Public

Connect IoT - Upload Recipe to Equipment Tutorial

Estimated time to read: 9 minutes

Overview

This tutorial shows how to configure Connect IoT to upload MES recipes directly to equipment. When an operator triggers an upload from the **Resource** page or the **Recipe** page, MES publishes a request on the message bus. The **Automation Controller** workflow receives it, performs the upload on the equipment side, and sends a reply — success or failure — back to MES.

This integration bridges the MES recipe repository with the equipment's internal recipe storage, eliminating manual file transfers and providing a traceable upload record per resource.

```
sequenceDiagram
    participant Operator
    participant MES
    participant MessageBus as Message Bus
    participant Controller as Automation Controller
    participant Equipment

    Operator->>MES: Trigger upload (Resource or Recipe page)
    MES->>MessageBus: Publish RecipeManagement.UploadRecipe
    MessageBus->>Controller: recipeUploadRequest fires
    Controller->>Equipment: Write recipe body to equipment
    Equipment-->>Controller: Upload result
    Controller->>MessageBus: recipeUploadReply (success / failure)
    MessageBus->>MES: Receive reply
    alt success
        MES->>MES: Update Last Upload Date & Version
        MES->>Operator: Upload successful
    else failure
        MES->>Operator: Show failure reason
    end
```

Use this approach when:

- **Recipe-driven equipment** — when equipment must be loaded with the correct recipe before production begins.
- **Audit and traceability** — when you need a record of which recipe version was uploaded to which resource and when. MES updates `Last Upload Date` and `Last Upload Version` on the Resource Recipe record after each successful upload.

Example

A cookie factory uses a baking oven that must receive the correct baking program before each production run. The recipe body is stored in MES and contains the temperature profile, timing, and allergen parameters. When the production engineer is ready, they select the recipe from the **Resource** page and click **Upload**. Connect IoT delivers the recipe to the oven and confirms success within seconds.

Prerequisites

Before configuring the upload workflow, confirm the following.

0. Load the tutorial masterdata

The tutorial uses pre-built masterdata packages that create the factory entities, the Connect IoT infrastructure, and the recipe records. Load them in order using **Master Data Packages**:

Order	File	What it creates
1	01-cookie-factory-base-masterdata.json	Enterprise, Site, Area, Resources (`EQ Baker-01`, `EQ Baker-02`), Flow, Steps, and Products
2	02-automation-masterdata.json	Automation Protocol (`EQ mqProt`), Driver Definition (`EQ mqDriverDef`), Controller (`EQ mqAutoController`), and Manager (`EQ recipe-upload-manager`)
3	03-recipe-masterdata.json	Recipes (`EQ BakingOreos`, `EQ BakingWafers`) and their Resource Recipe associations

After loading, the sections below describe how to verify the resulting configuration before proceeding to build the workflow.

1. Resource configuration

Open the **Resource** record and verify:

Property	Required value
`AutomationMode`	`Offline` or `Online`
`IsRecipeManagementEnabled`	`true`
`TrackResourceRecipes`	`true`
`IsUploadRecipeCapable`	`true`

The **Resource** entity type must also be enabled as a Connect IoT Integration Entity (see How To: Enable Connect IoT on an Entity Type) and the user performing the upload must be configured as an Integration User (see How To: Enable a Connect IoT Integration User).

2. Recipe configuration

The recipe must:

- Be **Active**, **Effective**, and not a Template.
- Have a non-empty **Body Source** and **Body** — the body is what gets sent to the equipment.
- Be associated with the Resource via a **Resource Recipe** record (Manage Resource Recipes).

3. Connect IoT Entities

The following entities must exist and be configured before starting:

Entity	Name used in this tutorial	Purpose
Automation Protocol	`EQ mqProt`	Defines the communication protocol (OPC-UA package used as message bus transport)
Automation Driver Definition	`EQ mqDriverDef`	Links the protocol to the `Resource` entity type
Automation Controller	`EQ mqAutoController`	Hosts the upload handling workflow
Automation Manager	`EQ recipe-upload-manager`	Runs the controller instance

Configuration

The upload integration uses Connect IoT's message bus to decouple MES from the equipment. MES publishes a request and waits for a reply — the **Automation Controller Workflow** owns the equipment interaction. This means the workflow can be adapted to any equipment protocol without changes to the MES side.

For the purposes of this tutorial we are not actually sending the recipe to the equipment — we are just using some Connect IoT tasks to simulate possible upload outcomes.

Component	Role
Recipe Upload Request task	Subscribes to `RecipeManagement.UploadRecipe` on the message bus. Fires when MES triggers an upload. Outputs the recipe name and other metadata.
Equipment interaction tasks	Perform the actual upload to the equipment (driver calls, timers, or custom code depending on integration).
Recipe Upload Reply task	Sends the reply back to MES — success or failure reason. MES uses this to update the upload record and display the result to the operator.

The following sub-sections detail the setup.

Create the Automation Controller workflow

Go to **Automation** > **Automation Controller**, select `EQ mqAutoController`, open the **Workflows** tab, and select **Add Workflow**.

Create a **Data Flow** workflow — e.g., named `upload handling` — with the following tasks:

1. Recipe Upload Request

Setting	Value
Task	Recipe Upload Request

Setting	Value
Auto Enable	`true`
Reply Timeout	`60000` ms

This task listens for upload requests published by MES. The `Reply Timeout` defines how long MES waits for a reply before showing a timeout error to the operator.

Outputs used downstream:

- `Success` → activate the next task
- `Recipe Name` → pass the recipe name to the routing logic

2. Expression Evaluator (routing)

Setting	Value
Task	Generic Expression Evaluator
Input	`recName` (String) — connected from `Recipe Name` output parameter in `Recipe Upload Request` task
Output `Success`	`recName == "EQ BakingOreos"` → Boolean
Output `Failure`	`recName != "EQ BakingOreos"` → Boolean
Evaluator Behavior	`EvaluateUntilTrue`

In a real implementation, replace this with the actual upload driver task. The expression evaluator is a mock that simulates routing — it returns `true` on the happy path and `false` for any recipe that would fail on the equipment.

3. Happy path — On Timer (mock upload delay)

Setting	Value
Task	On timer
Auto Enable	`false`
Interval	`5000` ms
Number of Occurrences	`1`

Activated by the `Success` output of the expression evaluator. Simulates a 5-second upload operation. In a real workflow, replace this with the driver task that writes the recipe body to the equipment.

Connect the **OnTimer** task `Success` output parameter with **Recipe Upload Reply** task's `Activate` input parameter.

4. Error path — Code Execution (mock error)

Setting	Value
Task	Custom Code

Activated by the `Failure` output of the expression evaluator. Emits an error object that is forwarded to **Recipe Upload Reply** task's `Failure Reason` input parameter.

In a real workflow, the error path is typically the error output of the upload driver task — no separate code task is needed.

Connect:

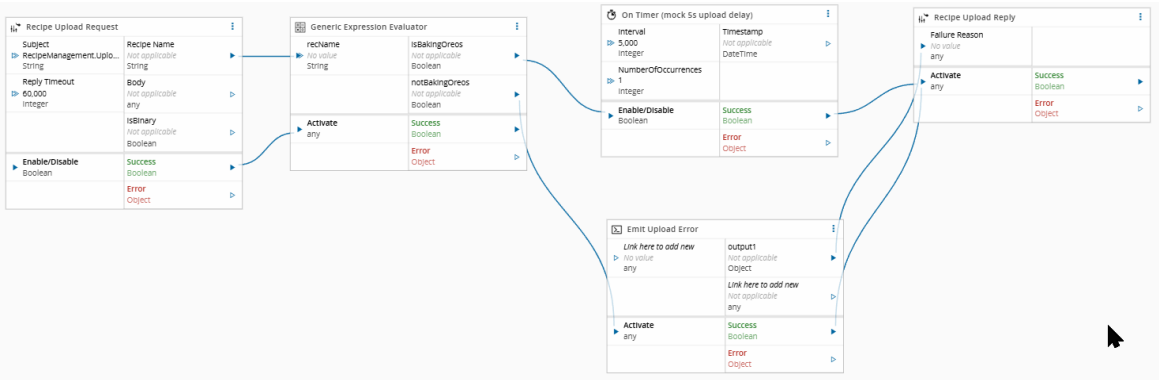
- **Code Execution** task's `Output1Out` parameter with **Recipe Upload Reply** task's `Failure Reason` input parameter.
- **Code Execution** task's `Success` output parameter with **Recipe Upload Reply** task's `Activate`.

5. Recipe Upload Reply

Setting	Value
Task	Recipe Upload Reply

Sends the result back to MES. If `Failure Reason` is connected and non-null, MES treats the upload as failed and displays the error message to the operator.

Full workflow diagram



Download and start the manager

After saving the workflow:

1. Open the **Automation Manager** `EQ recipe-upload-manager` record and click **Connect**.
2. Click **Download** to push the latest controller configuration to the manager runtime.
3. Click **Start** to bring the manager online.

Confirm the **Automation Controller Instance** shows `Running` status before attempting an upload.

Simulation

This simulation uses the mock workflow (expression evaluator + timer) to demonstrate the upload flow without real equipment.

Scenario 1: Happy path — successful upload

1. Navigate to `EQ Baker-01` or `EQ Baker-02` on the **Resource** page.
2. Open the **Resource Recipes** section.
3. Select the recipe `EQ BakingOreos` and click **Upload to Resource**.
4. MES publishes the request to the message bus. The controller workflow receives it and routes to the `onTimer` path.
5. After 5 seconds the timer fires and **Recipe Upload Reply** sends a success reply.
6. MES displays a success message and updates `Last Upload Date` and `Last Upload Version` on the **Resource Recipe** relation record.

Scenario 2: Error path — upload rejected by equipment

1. Select `EQ BakingWafers` (or any recipe whose name is not `EQ BakingOreos`) and click **Upload to Resource**.
2. The expression evaluator routes to the **Code Execution** error path.
3. **Recipe Upload Reply** task receives the error object as `Failure Reason` and sends a failure reply.
4. MES displays an error message. `Last Upload Date` is not updated.

Scenario 3: Timeout path — controller not running

1. Stop the `EQ recipe-upload-manager` (or disconnect it).
2. Attempt an upload from the **Resource** page.
3. MES waits for 60 seconds (the `Reply Timeout` configured on **Recipe Upload Request** task).
4. After the timeout, MES displays a timeout error. No update is made to the **Resource Recipe** relation record.

Scenario 4: Upload from the Recipe page

The same behavior applies when triggering from the **Recipe Page**, using **Upload to Resources**. Select the **Recipe**, pick one or more **Resources**, and confirm. Each **Resource** upload is processed independently through the same workflow. Re-run previous scenarios, but using this page.

Troubleshooting

Upload button is not visible

The **Upload** button only appears on resources that have `IsUploadRecipeCapable` property enabled. Verify this property is set on the **Resource** record. Also confirm the user has the `Resource.UploadRecipe` security feature assigned.

Upload never completes — always times out

Check that the **Automation Manager** is running and the controller instance status is `Running`. If the manager is down, the **Recipe Upload Request** task never fires and MES waits until the reply timeout expires.

Last Upload Date is not updated after a successful upload

`Last Upload Date` and `Last Upload Version` are only updated when `TrackResourceRecipes` property is enabled on the **Resource** and a **Resource Recipe** record already exists for the recipe/resource combination.

If the record is missing, add it via **Manage Resource Recipes** first.

Replacing the mock workflow with a real driver

In this tutorial the **Expression Evaluator** and **OnTimer** tasks are placeholders. For real equipment integration:

1. Replace the expression evaluator and timer with a driver task that writes the recipe body to the equipment (e.g., an OPC-UA Write command or a custom protocol command).
2. Connect the driver's success output to `RecipeUploadReply.Activate` input parameter.
3. Connect the driver's error output to `RecipeUploadReply.FailureReason` and then to `RecipeUploadReply.Activate` input parameters.
4. The Recipe Upload Request task outputs `Recipe Body`, `Recipe Version`, and `Recipe Name` — use these as inputs to your driver task.

Related Documentation

For reference details outside this tutorial, see these pages in the User Guide:

- Recipe Page
- Upload Recipe to Resource

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.