

Connect IoT - Control Flow Configuration Tutorial

11.1

January 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 - Control Flow Configuration Tutorial

The goal of this tutorial is to showcase the use of the control flow designer in constructing integrations. It will use a file raw integration. The goal of the integration will be whenever a new file is created with a content of Test followed by a number, the integration will create a number of files equal to the number specified in the content with the file name being the number and the content being success. If the content does not contain the keyword Test, log saying file ignored.

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 Protocol

The **Automation Protocol** will be quite simple, select the package `@criticalmanufacturing/connect-iot-driver-fileraw` in order to use the file raw protocol. Use the default configurations.

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.

Automation Driver Definition

The **Automation Driver Definition** will use the templates for the File Raw, as such it won't require any other configuration, besides depending on the previously created File Raw Protocol.

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.

Automation Controller

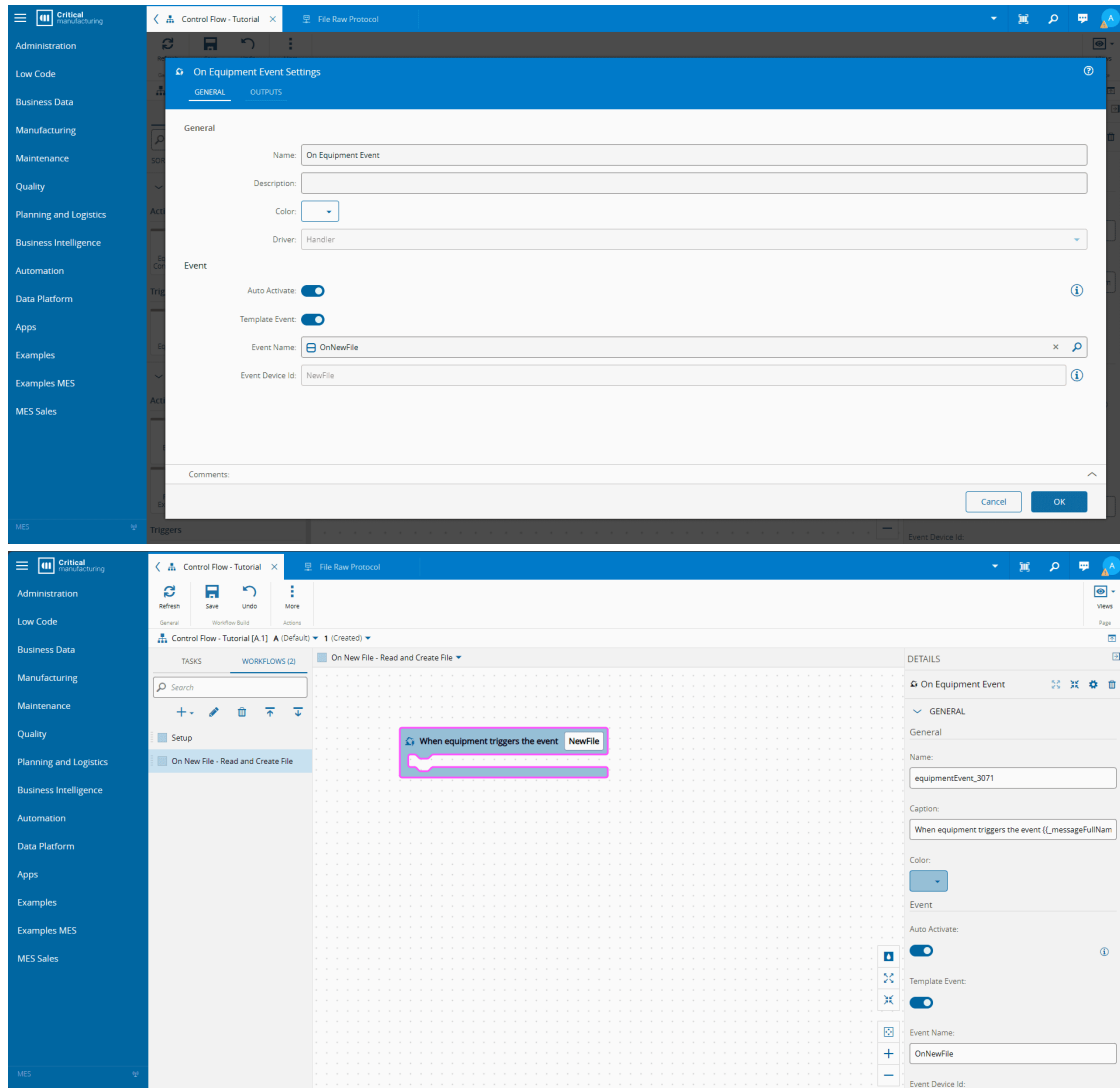
The **Automation Controller** will be of `Workflow Type` **Control Flow**. In the tutorial the entity type used was a of type `Site`. Use the `File Raw Driver Definition` previously created. In the `Tasks`, select also the `File Driver Tasks`.

Upon creating the Controller, the GUI will offer a default template for the driver created. This template will have the start cycle of the driver. In the Task `Equipment Configuration`, edit the `path` and provide the path for the watcher. For this tutorial the used path was `c:/temp/Tutorial`.

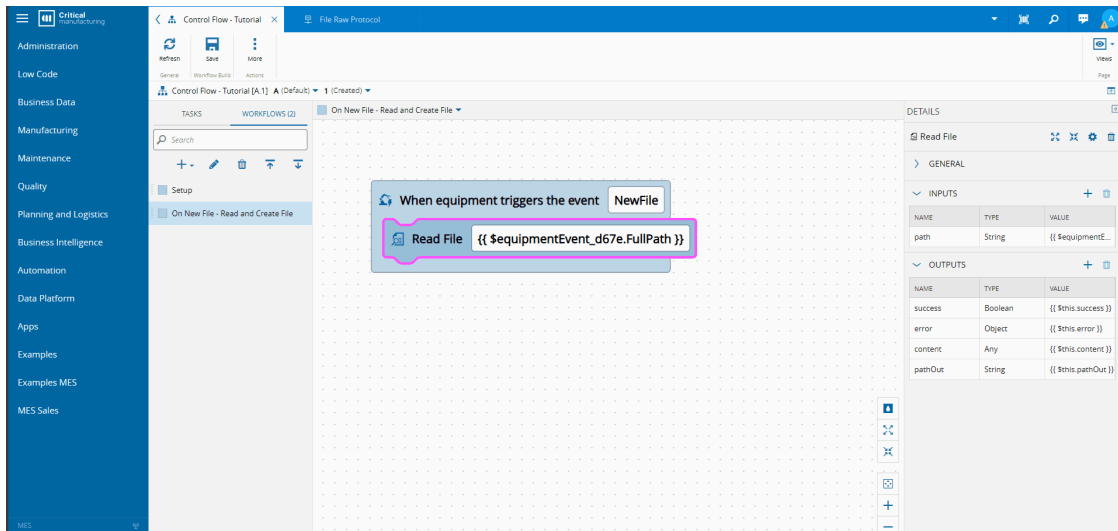
1. In the page left panel select `Workflows` and then `+`, select `Control Flow`. Edit the page and rename the page to a more friendly name, for example `On New File - Read and Create File`.
2. Drag and drop the following tasks:
 - `On Equipment Event`: to listen to the Event "OnNewFile", to be notified on a new file

- **Read File**: to read the file that was created
- **Condition**: to perform conditional operations
- **Create File**: to create a new file
- **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 **OnNewFile**.



4. Add the **Read File** Task as the next action. In **Control Flow**, all the context is available. You can either retrieve it from the general scope, or from a particular **Task**. In this case, the **Read Task** will require the full path of the **New File** detected.



Note

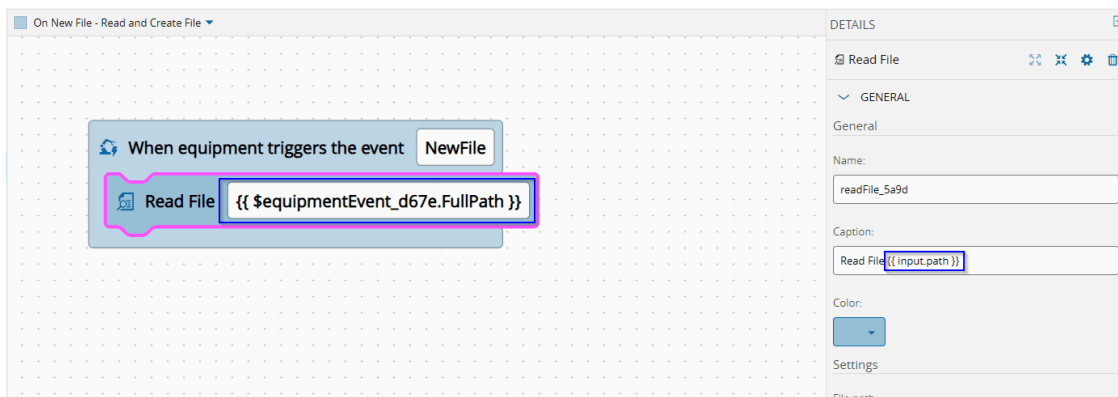
In **Control Flow** items in scope can be used using double curly braces and with dollar scope or dollar name of task, i.e. `{{\scope.Test.Name}}` or `{{\}task_1.Test.Name}}`. If the token is in the context of the task, as a setting or input, the token can be accessed directly for a setting or using `input.` for an input i.e. `{{path}}` or `{{input.path}}`.

Important

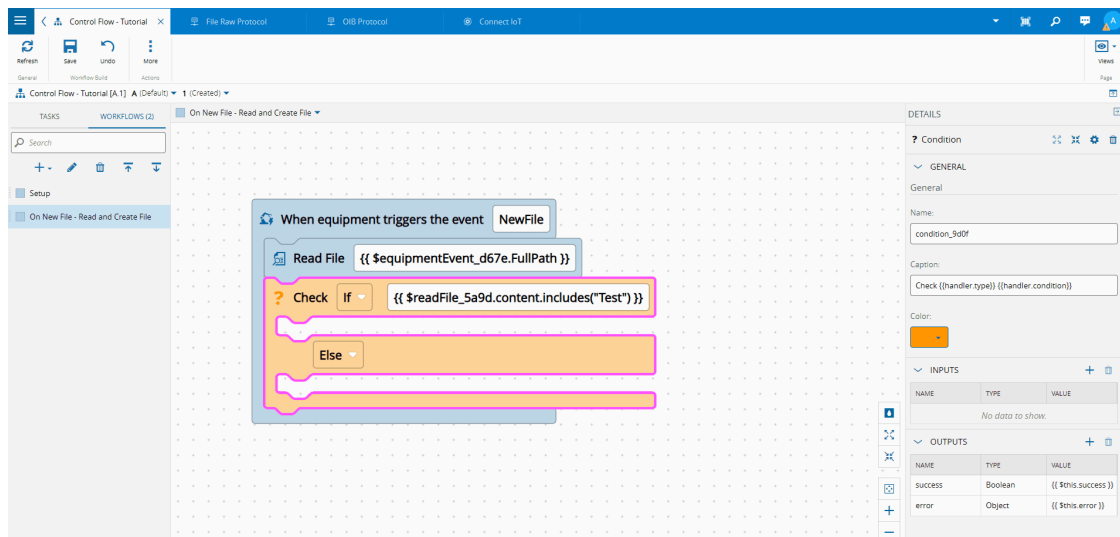
Only inputs and outputs are tokenizable. Currently, settings may not contain tokens, unless specific tasks mention that they can.

Note

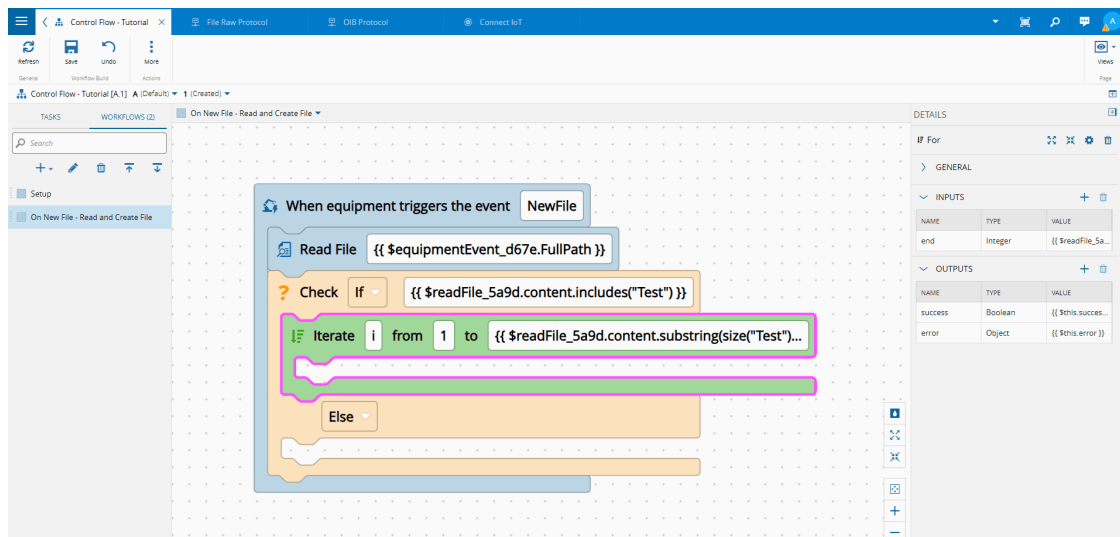
Notice the caption is rendering the content of the input, even though it is described as `{{ input.path }}`.



5. Add a condition task. If the read task content includes a value `Test`, it will be to process.



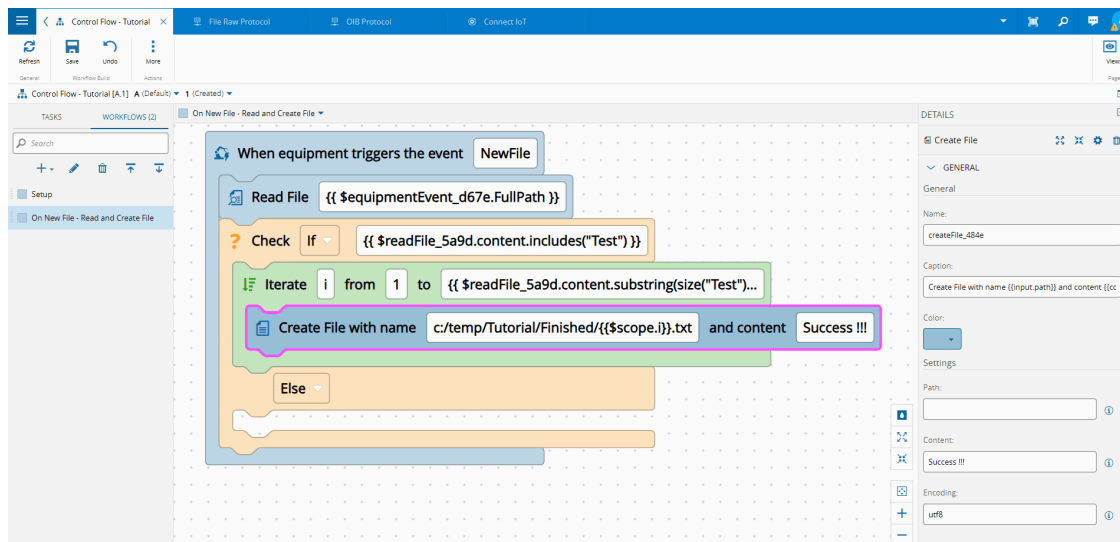
6. In the `If` condition add an iterator task. This task will iterate through the content of the file after the keyword `Test`. To use tokens, we will add an input `end`, with the token for the content of the read file task after the `Test`. The expression used will be `{{ $readFile_5a9d.content.substring(size("Test") [0]) }}`.



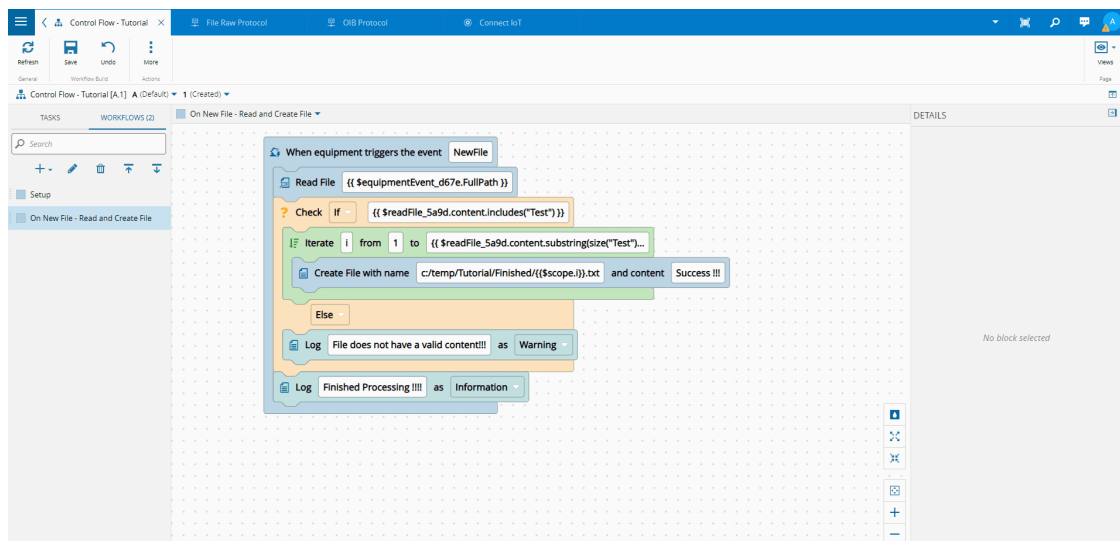
Note

In `Control Flow` token expressions allow for the use of more transformations. To apply transformation the engine applied is the `mathjs` engine. For indexes, the 0 starting point will be used.

7. In the `Iterator` task we can now add the `Create File`. The goal will be to create a file with name of the index and with content `Success !!!`. This file will be saved in the path `c:/temp/Tutorial/Finished`.



8. To finish our integration let us add a log message both for the else condition and for the end of the execution.



Execution

Let's now execute a run of our integration.

Drop a file with content `Test3` in the `c:/temp/Tutorial` folder:

```
2024-09-03 14:07:09.071 Info: [B04015] On New File - Read and Create File | Found package '@criticalmanufacturing/connect-lot-controller-engine-filedrivers-tasks' with version '11.1.0-202408201' in 'C:/vmp/Products/LOFT-CherryPicker/LOFT/Node_modules/@criticalmanufacturing/connect-lot-controller-engine-filedrivers-tasks'
2024-09-03 14:07:09.071 Info: [B04015] On New File - Read and Create File | Successfully updated task's outputs after received event 'undefined'.
2024-09-03 14:07:09.072 Info: [B04015] On New File - Read and Create File | Sending 'CreateFile' command. File to create: 'c:/temp/Tutorial/Finished/1.txt'; File content: Success !!!
2024-09-03 14:07:09.073 Info: [B04015] On New File - Read and Create File | Sending 'CreateFile' command. File to create: 'c:/temp/Tutorial/Finished/2.txt'; File content: Success !!!
2024-09-03 14:07:09.074 Info: [B04015] On New File - Read and Create File | Sending 'CreateFile' command. File to create: 'c:/temp/Tutorial/Finished/3.txt'; File content: Success !!!
2024-09-03 14:07:09.075 Info: [B04015] On New File - Read and Create File | Finished processing branches
2024-09-03 14:07:09.076 Info: [B04015] On New File - Read and Create File | Finished Processing!!!!
2024-09-03 14:07:09.077 Info: [B04015] On New File - Read and Create File | Finished Processing!!!!
```

Notice that we see a new file being detected, the iterator being activated and creating three new files.

If we drop an invalid file:

```
2024-09-03 14:08:11.415 info: Triggering custom event occurrence 'undefined'
  FileName=Test_ERROR.log || original=Test_ERROR.log
  RelativePath=Test_ERROR.log || original=Test_ERROR.log
  FullPath=c:\temp\tutorial\Test_ERROR.log || original=c:\temp\tutorial\Test_ERROR.log
  Size=5 || original=5
  IsFile=true || original=true
  IsDirectory=false || original=false
  AccessTime=Tue Sep 03 2024 14:08:01 GMT+0100 (Western European Summer Time) || original=2024-09-03T13:08:01.407Z
  CreationTime=Mon Sep 02 2024 17:53:55 GMT+0100 (Western European Summer Time) || original=2024-09-02T16:53:55.581Z
  UpdateTime=Tue Sep 03 2024 14:08:01 GMT+0100 (Western European Summer Time) || original=2024-09-03T13:08:01.407Z
2024-09-03 14:08:11.423 info: Sending command 'ReadFile' to device. Command was sent by a controller task.
2024-09-03 14:08:11.418 info: [9f111466]On New File - Read and Create File|17253527831238691[equipmentEvent] Received a 'undefined' event.
2024-09-03 14:08:11.419 info: [9f111466]On New File - Read and Create File|17253527831238691[equipmentEvent] Successfully updated task's outputs after received event 'undefined'.
2024-09-03 14:08:11.423 info: [e9ba81b8]On New File - Read and Create File|17253528881448782[readFile] Sending ReadFile command. File to read: c:\temp\tutorial\Test_ERROR.log
2024-09-03 14:08:11.423 info: Sending to 'Handler' a message of type 'connect.iot.driver.fileBased.executeCommand'
2024-09-03 14:08:11.429 info: [e9ba81b8]On New File - Read and Create File|17253554424619187[condition] Validating which branch can be executed
2024-09-03 14:08:11.433 warn: [cc9c63a0]On New File - Read and Create File|172536694081010101[logMessage] File does not have a valid content!!!
2024-09-03 14:08:11.434 info: [e9ba81b8]On New File - Read and Create File|17253554424619187[condition] Finished processing branches
2024-09-03 14:08:11.435 info: [e9ba81b8]On New File - Read and Create File|172536697560910126[logMessage] Finished Processing !!!!
```

It will not create any file and will log a warning notifying the user that it was not processed.

This tutorial tries to illustrate a simple use case for the use of control flow and how dynamic information can be propagated.



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.