Automation Business Scenario - Basic Scenario

11.3

April 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.

Automation Business Scenario - Basic Scenario

Estimated time to read: 11 minutes

The business scenarios is a framework and execution engine that allows the user to construct, via metadata and through user interaction, complex flows in order to perform actions in the MES system.

This document will guide you through the process of creating a scenario for a particular use case.

Overview

In this tutorial the goal will be to create a scenario that will query the user for the Automation Managers he wishes to Deploy.

> **Info**
>
> For this tutorial, only the MES UI will be used, nevertheless it is strongly advised for the user to use the CM CLI. The `CLI` allows generating a package for IoT and to create customization packages that hold the `Automation Business Scenarios`. Using the visual studio code extension Automation Business Scenarios Renderer, is also helpful as it provides a language formatter for the Business Scenario Structure and a diagram renderer in mermaid chart of the scenario.

Building a Scenario

The first step in building a scenario is defining the decision tree for the user. For this tutorial the end goal is to have a list of **Automation Managers**, that are of type `Automatic Deploy` and are not in the state `Ready` (which means they can be deployed), and perform the deployment.

The user may interact with the scenario in two fashions, he can provide a list of comma separated **Automation Managers** (this will be a `Manual` mode) or he can choose the managers from a list (the `Interactive` mode).

Defining a Scope

One of the important parts of a scenario is defining in what context or scope it should be available to execute. For this tutorial, the scope is defined when the user is in the page listing the **Automation Managers**, so the scope will be `Entity/AutomationManager`.

Validating the User

A condition to be able to deploy the **Automation Manager** is for the user to be an `Integration User`. Therefore, the first step must be to validate the user and store the user. The user will be needed later in order to execute the deploy.

The first step must be defined as the value for the `start`. The step will be of type Script and will execute a simple script to check if the user is an integration user. It will also store the user using the `resultKey` in the key `selectedUser`.

```
{
```

```
    "metadata": {
        "start": "CheckIfUserIsIntegrationUser",
        "steps": [
            {
                "name": "CheckIfUserIsIntegrationUser",
                "type": "Script",
                "resultKey": "selectedUser",
                "settings": {
                    "script": [
                        "if(!this.securityService.user.IsIntegrationUser) {",
                        "   throw new Error('This scenario can only be executed by a User that is
an Integration User');",
                        "}",
                        "this.securityService.user"
                    ]
                },
                "next": "Mode"
            }
        ]
    }
}
```

Interaction Modes

One of the features that the tutorial must support is the two modes, one that is the `Manual` mode where the user specifies a list of comma separated **Automation Managers** and an `Interactive` mode that prompts the user to choose a manager from a list. The scenario will ask a question of the user with two options: `Manual`, `Interactive`, using the step type Question. The user will choose one of the options, by choosing an option the scenario can introduce forks in the flow, by using the step type Condition.

> **Note**
>
> In the step type Condition, the `next` key will act as the default result if no condition is matched. In a case, where the result should match always one of the options, the user should terminate the scenario with an error message.

```
{
    "metadata": {
        (...)
        "steps": [
            (...)
            {
                "name": "Mode",
                "type": "Question",
                "resultKey": "selectionMode",
                "settings": {
                    "message": "Do you wish to have a manual selection with ',' separated Manager
Names or the interactive mode?",
                    "dataType": "Enum",
                    "settings": {
                        "enumValues": [
                            "Manual",
                            "Interactive"
                        ]
                    },
                    "defaultValue": "Interactive"
                },
                "next": "ModeCondition"
            },
            {
                "name": "ModeCondition",
                "type": "Condition",
                "settings": {
                    "condition": {
                        "selectionMode == 'Manual'": "CommaSeparatedManagers",
                        "selectionMode == 'Interactive'": "SelectManagerToDeploy"
                    }
```

```
                },
                "next": "Error"
            },
            {
                "name": "Error",
                "type": "Script",
                "settings": {
                    "script": [
                        "throw new Error('Something went wrong with this scenario.')"
                    ]
                },
                "next": ""
            }
        ]
    }
}
```

Handling Manual Mode

When the user chooses the `Manual` mode, the scenario should ask a question and extract the comma separated list of managers. For the `Manual` mode this will be the end of the scenario.

```
{
    "metadata": {
        (...)
        "steps": [
            (...)
            {
                "name": "CommaSeparatedManagers",
                "type": "Question",
                "resultKey": "managersToDeployComma",
                "settings": {
                    "message": "Please provide a list of Manager Names, separated by ','.",
                    "dataType": "String"
                },
                "next": ""
            }
}
```

Handling Interaction Mode

When the user chooses the `Interactive` mode the scenario should give the user a list of possible managers to select and add it to the selected managers list. The user may choose several **Automation Managers**.

> **Note**
>
> Notice the reference to `${script(....ts)}` the use of the `CLI` allows for the user to have more complex scripts in different files. The scripts will then be converted to `Base64` when the customization package is run with `cmf pack`.

Let's create a script with a `Query Object`. This query will retrieve all **Automation Managers** that were not already selected, that are not `Terminated`, that have deployment mode `AutomaticDeploy` and are not in state `Ready`. After each reply, the result is stored.

Here is the script defined above as `scripts/mass-deploy/managers_to_deploy.ts`:

scripts/mass-deploy/managers_to_deploy.ts

```
const filterCollection = new Cmf.Foundation.BusinessObjects.QueryObject.FilterCollection();

// Selected Manager Filters
if (this.answers.selectedManagers != null && this.answers.selectedManagers.length > 0) {
    this.answers.selectedManagers.forEach(selectedManager => {
```

```
        const filterManagerAlreadySelected = new
Cmf.Foundation.BusinessObjects.QueryObject.Filter();
        filterManagerAlreadySelected.Name = "Name";
        filterManagerAlreadySelected.ObjectName = "AutomationManager";
        filterManagerAlreadySelected.ObjectAlias = "AutomationManager_1";
        filterManagerAlreadySelected.Operator = Cmf.Foundation.Common.FieldOperator.IsNotEqualTo;
        filterManagerAlreadySelected.Value = selectedManager.Name;
        filterManagerAlreadySelected.LogicalOperator =
Cmf.Foundation.Common.LogicalOperator.Nothing;
        filterManagerAlreadySelected.FilterType =
Cmf.Foundation.BusinessObjects.QueryObject.Enums.FilterType.Normal;
        filterCollection.push(filterManagerAlreadySelected);
    });
}

// Filter filter_0
const filter_0 = new Cmf.Foundation.BusinessObjects.QueryObject.Filter();

filter_0.Name = "UniversalState";
filter_0.ObjectName = "AutomationManager";
filter_0.ObjectAlias = "AutomationManager_1";
filter_0.Operator = Cmf.Foundation.Common.FieldOperator.IsNotEqualTo;
filter_0.Value = Cmf.Foundation.Common.Base.UniversalState.Terminated;
filter_0.LogicalOperator = Cmf.Foundation.Common.LogicalOperator.AND;
filter_0.FilterType = Cmf.Foundation.BusinessObjects.QueryObject.Enums.FilterType.Normal;

filterCollection.push(filter_0);

// Filter filter_1
const filter_1 = new Cmf.Foundation.BusinessObjects.QueryObject.Filter();

filter_1.Name = "DeploymentMode";
filter_1.ObjectName = "AutomationManager";
filter_1.ObjectAlias = "AutomationManager_1";
filter_1.Operator = Cmf.Foundation.Common.FieldOperator.IsEqualTo;
filter_1.Value = Cmf.Foundation.BusinessObjects.AutomationManagerDeploymentMode.AutomaticDeploy;
filter_1.LogicalOperator = Cmf.Foundation.Common.LogicalOperator.AND;
filter_1.FilterType = Cmf.Foundation.BusinessObjects.QueryObject.Enums.FilterType.Normal;

filterCollection.push(filter_1);

// Filter filter_2
const filter_2 = new Cmf.Foundation.BusinessObjects.QueryObject.Filter();

filter_2.Name = "DeploymentState";
filter_2.ObjectName = "AutomationManager";
filter_2.ObjectAlias = "AutomationManager_1";
filter_2.Operator = Cmf.Foundation.Common.FieldOperator.IsNotEqualTo;
filter_2.Value = Cmf.Foundation.BusinessObjects.AutomationManagerDeploymentState.Ready;
filter_2.LogicalOperator = Cmf.Foundation.Common.LogicalOperator.AND;
filter_2.FilterType = Cmf.Foundation.BusinessObjects.QueryObject.Enums.FilterType.Normal;

filterCollection.push(filter_2);

const fieldCollection = new Cmf.Foundation.BusinessObjects.QueryObject.FieldCollection();

// Field field_0
const field_0 = new Cmf.Foundation.BusinessObjects.QueryObject.Field();

field_0.Alias = "Id";
field_0.ObjectName = "AutomationManager";
field_0.ObjectAlias = "AutomationManager_1";
field_0.IsUserAttribute = false;
field_0.Name = "Id";
field_0.Position = 0;
field_0.Sort = Cmf.Foundation.Common.FieldSort.NoSort;

// Field field_1
const field_1 = new Cmf.Foundation.BusinessObjects.QueryObject.Field();

field_1.Alias = "Name";
field_1.ObjectName = "AutomationManager";
field_1.ObjectAlias = "AutomationManager_1";
```

```
field_1.IsUserAttribute = false;
field_1.Name = "Name";
field_1.Position = 1;
field_1.Sort = Cmf.Foundation.Common.FieldSort.NoSort;

fieldCollection.push(field_0);
fieldCollection.push(field_1);

const query = new Cmf.Foundation.BusinessObjects.QueryObject.QueryObject();

query.Description = "With Automatic Deployment Mode and State different from Ready";
query.EntityTypeName = "AutomationManager";
query.Name = "GetAllAutomationManagersForMassDeployment";
query.Query = new Cmf.Foundation.BusinessObjects.QueryObject.Query();
query.Query.Distinct = false;
query.Query.Filters = filterCollection;
query.Query.Fields = fieldCollection;

query;
```

In this case the cycle of questions is done by self referencing, as we can see in the step `Iterator`. The step type Foreach can be used to achieve a similar mechanism.

```
{
    "metadata": {
        (...)
        "steps": [
            (...)
            {
                "name": "SelectManagerToDeploy",
                "type": "Question",
                "resultKey": "currentSelectedManager",
                "settings": {
                    "message": "Please select an Automation Manager to Deploy:",
                    "dataType": "FindEntity",
                    "settings": {
                        "query": "${script(./scripts/mass-deploy/managers_to_deploy.ts)}"
                    }
                },
                "next": "PushManager"
            },
            {
                "name": "PushManager",
                "type": "Script",
                "settings": {
                    "script": [
                        "if (this.answers.selectedManagers == null) {",
                        "   this.answers.selectedManagers = [];",
                        "}",
                        "this.answers.selectedManagers.push(this.answers.currentSelectedManager);"
                    ]
                },
                "next": "DoYouWishToIterate"
            },
            {
                "name": "DoYouWishToIterate",
                "type": "Question",
                "resultKey": "isToIterate",
                "settings": {
                    "message": "Do you wish to deploy more Automation Managers?",
                    "dataType": "Boolean"
                },
                "next": "Iterator"
            },
            {
                "name": "Iterator",
                "type": "Condition",
                "settings": {
                    "condition": {
                        "isToIterate == true": "SelectManagerToDeploy",
                        "isToIterate == false": ""
```

```
                    }
                },
                "next": "Error"
            }
}
```

End

There are three different `end` possibilities: `MasterData`, `Script`, `Custom`. The `MasterData`, generates a `MasterData` package, the `Script` will execute a step of type Script and `Custom` will execute an arbitrary step.

For this tutorial, the `end` will be of type Script. The script will retrieve all the chosen **Automation Managers** and change the state of the **Automation Manager** to `Ready`. Also, the change requires specifying the user that is performing this change.

In order to perform this update of the entity, the script will invoke the service `FullUpdateObjects`, that will allow to edit the properties of a set of entities of the same type.

Notice how the scripts leverages questions that have been made throughout the scenario, like the user that is running the scenario and the managers that were chosen. All answers are storedi in the `answers` object.

Here is the script defined above as `scripts/mass-deploy/mass_deploy.ts`:

scripts/mass-deploy/mass_deploy.ts

```
(async () => {
    const input =
        new
Cmf.Foundation.BusinessOrchestration.GenericServiceManagement.InputObjects.FullUpdateObjectsInput();

    input.Objects = new Map();

    // Parse Managers from Manual mode
    if (this.answers.managersToDeployComma && this.answers.managersToDeployComma !== "") {
        this.answers.selectedManagers =
this.answers.managersToDeployComma.split(",").map(managerName => {
            const manager = new Cmf.Foundation.BusinessObjects.AutomationManager();
            manager.Name = managerName;

            return manager;
        });
    }

    // Iterate each Manager and change the state to Ready and add the user
    for (let automationManager of this.answers.selectedManagers) {

        const inputObject = new
Cmf.Foundation.BusinessOrchestration.GenericServiceManagement.InputObjects.GetObjectByNameInput();

        inputObject.Type = automationManager["$type"] ??
"Cmf.Foundation.BusinessObjects.AutomationManager, Cmf.Foundation.BusinessObjects";
        inputObject.Name = automationManager.Name;
        automationManager = (await this.System.call(inputObject)).Instance;

        const deploymentConfiguration = JSON.parse(automationManager.DeploymentConfiguration ?? "
{}");
        deploymentConfiguration["UserName"] = this.answers.selectedUser.UserName;
        deploymentConfiguration["UserAccount"] = this.answers.selectedUser.UserAccount;

        automationManager.DeploymentConfiguration = JSON.stringify(deploymentConfiguration, null,
"\t");
        automationManager.DeploymentState =
Cmf.Foundation.BusinessObjects.AutomationManagerDeploymentState.Ready;

        input.Objects.set(automationManager, new
Cmf.Foundation.BusinessOrchestration.FullUpdateParameters());
```

```
    }

    await this.System.call(input);
})();
```

```
{
    "end": "DeployManagers",
    "metadata": {
        (...)
        "steps": [
            (...)
            {
                "name": "DeployManagers",
                "type": "Script",
                "settings": {
                    "script": "${script(./scripts/mass-deploy/mass_deploy.ts)}"
                },
                "next": ""
            }
}
```

Conclusion

This tutorial shows how easy it can be to automate interactions with the `MES`, by building simple step based scenarios.

Scenario Diagram

Note

This diagram is rendered using the Automation Business Scenarios Renderer.

```
graph TD
classDef startClass fill: #007ac9, color:#000000;
classDef finallyClass fill: #50b450, color:#000000;
classDef endClass fill: #3b8b3b, color:#000000;
    CheckIfUserIsIntegrationUser["Script:
CheckIfUserIsIntegrationUser
(selectedUser)"] --> Mode
    Mode["Question:
Mode
(selectionMode)"] --> ModeCondition
    ModeCondition["Condition:
ModeCondition"] -->
|"selectionMode == 'Manual'"|CommaSeparatedManagers
    ModeCondition["Condition:
ModeCondition"] -->
|"selectionMode == 'Interactive'"|SelectManagerToDeploy
    SelectManagerToDeploy["Question:
SelectManagerToDeploy
(currentSelectedManager)"] --> PushManager
    PushManager["Script:
PushManager"] --> DoYouWishToIterate
    DoYouWishToIterate["Question:
DoYouWishToIterate
(isToIterate)"] --> Iterator
    Iterator["Condition:
Iterator"] -->
|"isToIterate == true"|SelectManagerToDeploy
    Iterator["Condition:
Iterator"] -->
|"isToIterate == false"|StepExecution["End Step Execution"]:::startClass
    StartStep["Start Step"]:::startClass  --> CheckIfUserIsIntegrationUser
    EndStep["End Step"]:::endClass --> DeployManagers
```

Full Scenario (JSON representation):

The json representation showed bellow is the that will be used with the `CLI` package for business scenarios.

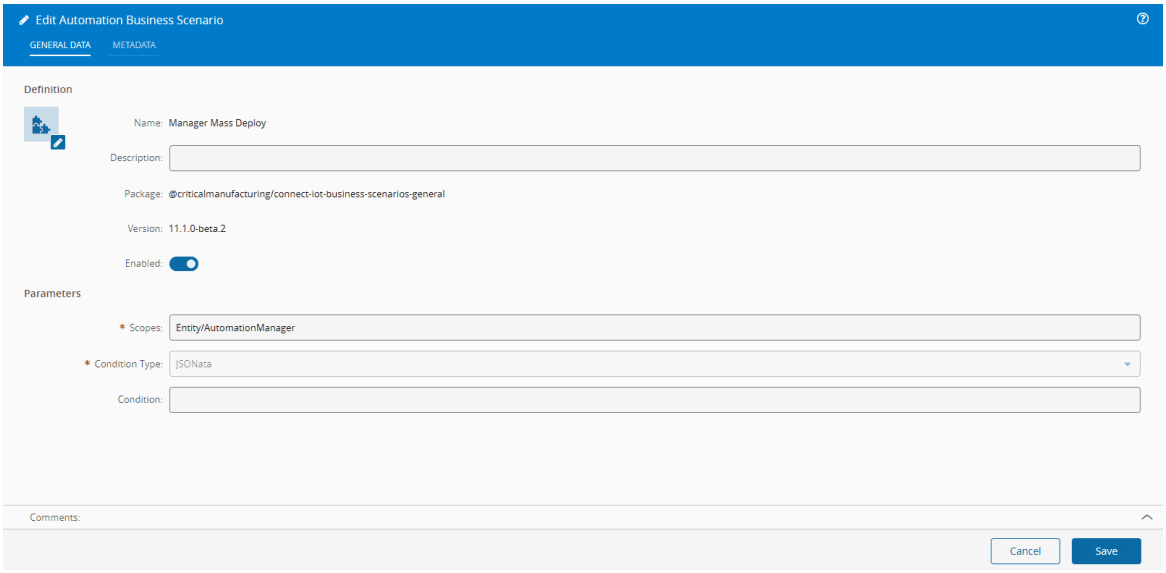
```
{
    "name": "Manager Mass Deploy",
    "description": "Automation Manager Mass Deploy",
    "scopes": "Entity/AutomationManager",
    "conditionType": "JSONata",
    "condition": "",
    "metadata": {
        "start": "CheckIfUserIsIntegrationUser",
        "resultType": "Script",
        "end": "DeployManagers",
        "steps": [
            {
                "name": "CheckIfUserIsIntegrationUser",
                "type": "Script",
                "resultKey": "selectedUser",
                "settings": {
                    "script": [
                        "if(!this.securityService.user.IsIntegrationUser) {",
                        "    throw new Error('This scenario can only be executed by a User that is
an Integration User');",
                        "}",
                        "this.securityService.user"
                    ]
                },
                "next": "Mode"
            },
            {
                "name": "Mode",
                "type": "Question",
                "resultKey": "selectionMode",
                "settings": {
                    "message": "Do you wish to have a manual selection with ',' separated Manager
Names or the interactive mode?",
                    "dataType": "Enum",
                    "settings": {
                        "enumValues": [
                            "Manual",
                            "Interactive"
                        ]
                    },
                    "defaultValue": "Interactive"
                },
                "next": "ModeCondition"
            },
            {
                "name": "ModeCondition",
                "type": "Condition",
                "settings": {
                    "condition": {
                        "selectionMode == 'Manual'": "CommaSeparatedManagers",
                        "selectionMode == 'Interactive'": "SelectManagerToDeploy"
                    }
                },
                "next": "Error"
            },
            {
                "name": "CommaSeparatedManagers",
                "type": "Question",
                "resultKey": "managersToDeployComma",
                "settings": {
                    "message": "Please provide a list of Manager Names, separated by ','.",
                    "dataType": "String"
                },
                "next": ""
```

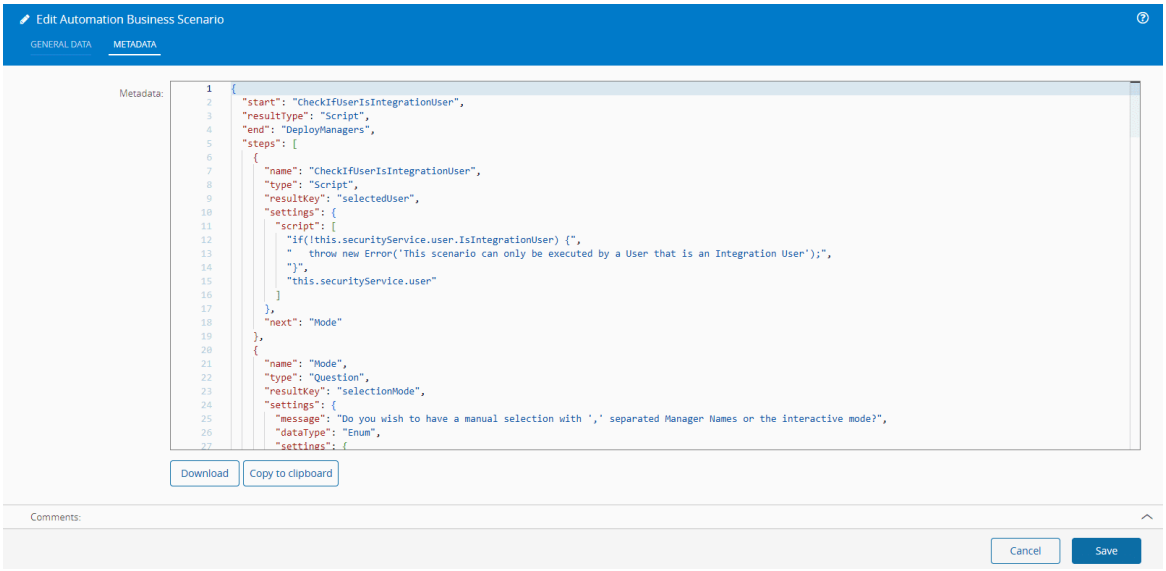
```
            },
            {
                "name": "SelectManagerToDeploy",
                "type": "Question",
                "resultKey": "currentSelectedManager",
                "settings": {
                    "message": "Please select an Automation Manager to Deploy:",
                    "dataType": "FindEntity",
                    "settings": {
                        "query": "${script(./scripts/mass-deploy/managers_to_deploy.ts)}"
                    }
                },
                "next": "PushManager"
            },
            {
                "name": "PushManager",
                "type": "Script",
                "settings": {
                    "script": [
                        "if (this.answers.selectedManagers == null) {",
                        "   this.answers.selectedManagers = [];",
                        "}",
                        "this.answers.selectedManagers.push(this.answers.currentSelectedManager);"
                    ]
                },
                "next": "DoYouWishToIterate"
            },
            {
                "name": "DoYouWishToIterate",
                "type": "Question",
                "resultKey": "isToIterate",
                "settings": {
                    "message": "Do you wish to deploy more Automation Managers?",
                    "dataType": "Boolean"
                },
                "next": "Iterator"
            },
            {
                "name": "Iterator",
                "type": "Condition",
                "settings": {
                    "condition": {
                        "isToIterate == true": "SelectManagerToDeploy",
                        "isToIterate == false": ""
                    }
                },
                "next": "Error"
            },
            {
                "name": "DeployManagers",
                "type": "Script",
                "settings": {
                    "script": "${script(./scripts/mass-deploy/mass_deploy.ts)}"
                },
                "next": ""
            },
            {
                "name": "Error",
                "type": "Script",
                "settings": {
                    "script": [
                        "throw new Error('Something went wrong with this scenario.')"
                    ]
                },
                "next": ""
            }
        ]
    }
}
```

Full Scenario (UI representation):

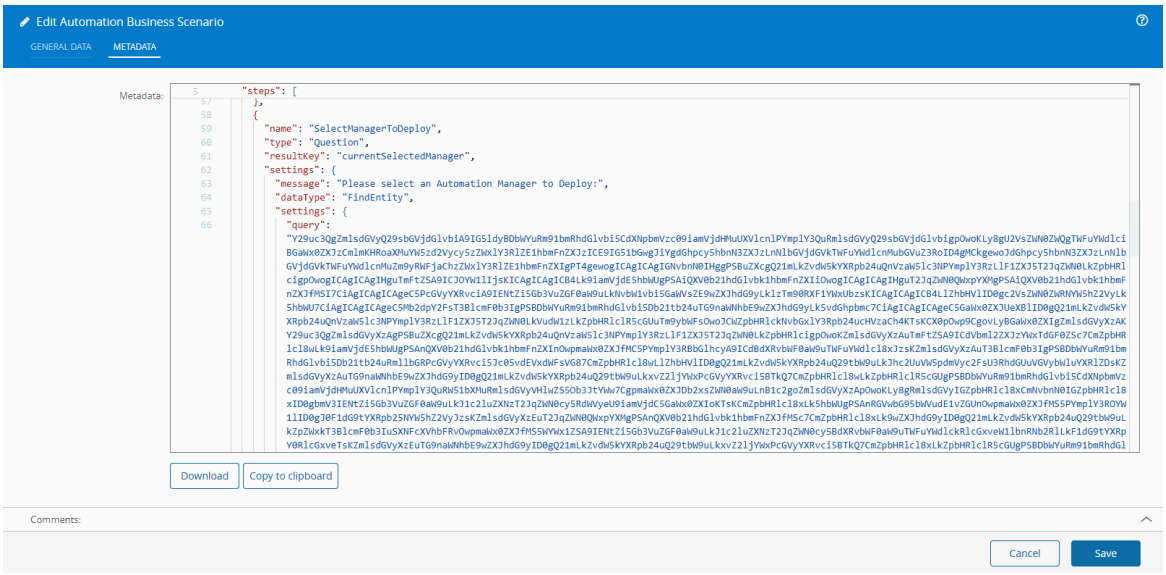
In the UI the representation of the scenario can be seen and edited as follows.

> **Info**
>
> Changes in the scenario will immediately impact the scenario execution in the next runs.





Notice also how when the script is uploaded to the system the script location references have been overriden with a transpilation of typescript to javascript and a conversion to base64, this process is automatic when using the `CLI`.


Edit Automation Business Scenario
GENERAL DATA
METADATA
Metadata:
"steps": [
},
{
"name": "SelectManagerToDeploy",
"type": "Question",
"resultKey": "currentSelectedManager",
"settings": {
"message": "Please select an Automation Manager to Deploy:",
"dataType": "FindEntity",
"settings": {
"query":
Download
Copy to clipboard
Comments:
Cancel
Save

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.