



**BUILDING BLOCKS / Actions**

# Contents

- [BUILDING BLOCKS / Actions / Business Rule action / DMN Business Rule action](#)
- [BUILDING BLOCKS / Actions / Send data to user interface](#)
- [BUILDING BLOCKS / Actions / Upload File action](#)
- [BUILDING BLOCKS / Actions / Start Subprocess action](#)
  - [Configuring a Start Subprocess action](#)
    - [Action Edit](#)
    - [Back in steps](#)
    - [Parameters](#)
    - [Data to send](#)
  - [Example](#)
- [BUILDING BLOCKS / Actions / Append Params to Parent Process](#)

## BUILDING BLOCKS / Actions / Business Rule action / DMN Business Rule action

For a brief introduction to

The fallback content to display on prerendering  
, check the following section:

» [Intro to DMN](#)

### Creating a DMN Business Rule action

To create and attach a DMN

The fallback content to display on prerendering  
action to a task

The fallback content to display on prerendering  
, you must do the following:

1. Open

The fallback content to display on prerendering  
and go to

The fallback content to display on prerendering

.

2. Select your process from the list and click **Edit process**.

3. Select a **task node** then click the **edit button** (the key icon) - this will open the node configuration menu.

4. In the opened menu, go to the

The fallback content to display on prerendering  
tab then click the "+" button.

5. From the dropdown menu choose the action type - **Business Rule**.

6. In the **Language** dropdown menu, select **DMN**.



## Using a DMN Business Rule action

We have the following scenario, a bank needs to perform client identification tasks/actions. This action can be defined as a

The fallback content to display on prerendering inside a

The fallback content to display on prerendering process using

The fallback content to display on prerendering

.

A business person or specialist can use DMN to design this business rule, without having to go deep into technical definitions.

Here is an example of an **MVEL** script - defined as a business rule action inside a **Service Task** node:

```
closedClientType = ["PF_CLOSED", "PF_SPECIAL", "PF_ABC",  
"PJ_CLOSED"];  
clientType =  
input.get("application").get("client").get("clientType");  
if (closedClientType.contains(clientType)) {  
    alertTitle = "Customer no longer with the bank";  
    alertDescription = "Hey! This person was a client  
before. For a new account modify the CIF.";  
    output.put("applications", {"client": {"alertTitle":  
alertTitle, "alertDescription": alertDescription}});  
}
```

The previous example could be easily transformed into a DMN Business Rule action - represented by the decision table:

Decision table <span>Hit Policy: Unique</span>				
	When		And	
	application.client.clientType	+	alert_title	+
	string		string	Annotations
1	IN ("PF_CLOSED", "PF_SPECIAL", "PF_ABC", "PJ_CLOSED")		Customer no longer with the bank.	Hey! This person was a client before. For a new account, modify the CIF
2	-			
+	-			

In the example above we used FEEL expression language in order to write the rules that should be met in order for the output to happen. FEEL defines a syntax for expressing conditions that input data should be evaluated against.

**Input** - In the example above we used as inputs the type of clients (inside a bank) using the `application.client` key

**Output** - In the example above we used as inputs the type of clients (inside a bank) using the `application.client` key

DMN also defines an XML schema that allows DMN models to be used across multiple DMN authoring platforms. The following output is the XML source of the decision table example from the previous section:

```
// Decision Table XML source
<?xml version="1.0" encoding="UTF-8"?>
<definitions
xmlns="https://www.omg.org/spec/DMN/20191111/MODEL/"
xmlns:biodi="http://bpmn.io/schema/dmn/biodi/2.0"
id="Definitions_06nober" name="DRD"
namespace="http://camunda.org/schema/1.0/dmn"
exporter="Camunda Modeler" exporterVersion="5.0.0">
  <decision id="closed_CIF" name="Decision table">
    <decisionTable id="decisionTable_1">
      <input id="input_1"
label="application.client.clientType" biodi:width="277">
        <inputExpression id="inputExpression_1"
typeRef="string">
          <text></text>
        </inputExpression>
      </input>
      <output id="output_1" label="alert_title"
typeRef="string" />
      <output id="OutputClause_043h9fw"
label="alert_description" typeRef="string" />
      <rule id="DecisionRule_10bh1zx">
        <inputEntry id="UnaryTests_0a6rf6l">
          <text>IN ("PF_CLOSED", "PF_SPECIAL", "PF_ABC",
"PJ_CLOSED")</text>
        </inputEntry>
      </rule>
    </decisionTable>
  </decision>
</definitions>
```

```
<outputEntry id="LiteralExpression_0xszo8x">
  <text>Customer no longer with the bank.</text>
</outputEntry>
<outputEntry id="LiteralExpression_0l2bioo">
  <text>Hey! This person was a client before. For a
new account, modify the CIF</text>
</outputEntry>
</rule>
<rule id="DecisionRule_1jj1rv2">
  <inputEntry id="UnaryTests_0cf2e91">
    <text></text>
  </inputEntry>
  <outputEntry id="LiteralExpression_1b9jkr4">
    <text></text>
  </outputEntry>
  <outputEntry id="LiteralExpression_12hua2f">
    <text></text>
  </outputEntry>
</rule>
</decisionTable>
</decision>
</definitions>
```

You can use this XML example with FLOWX Designer, adding it to a Business Rule Action - using an MVEL script. Then you can switch to DMN if you need to generate a graphical representation of the model.

**Was this page helpful?**

# BUILDING BLOCKS / Actions / Send data to user interface

## ! INFO

**What is it?** Send data to user interface

The fallback content to display on prerendering is based on Server-Sent Events (SSE), a web technology that enables servers to push real-time updates or events to clients over a single, long-lived HTTP connection. It provides a unidirectional communication channel from the server to the client, allowing the server to send updates to the client without the need for the client to continuously make requests.

**Why is it useful?** It provides real-time updates and communication between the

The fallback content to display on prerendering and the frontend application.

## Configuring a Send data to user interface action

Multiple options are available for this type of action and can be configured via the

The fallback content to display on prerendering

. To configure a Send data to user interface, use the **Actions** tab at the **task node level**, which has the following configuration options:

- **Action Edit**



- [Back in steps \(for Manual actions\)](#)
- [Parameters](#)
- [Data to send \(for Manual actions\)](#)

## Action Edit

- **Name** - used internally to make a distinction between different actions on nodes in the process. We recommend defining an action naming standard to be able to quickly find the process actions
- **Order** - if multiple actions are defined on the same node, the running order should be set using this option
- **Timer expression** - it can be used if a delay is required on that action. The format used for this is [ISO 8601 duration format](#) (for example, a delay of 30 seconds will be set up as `PT30S`)
- **Action type** - should be set to Send data to user interface
- **Trigger type** (options are Automatic/Manual) - choose if this action should be triggered automatically (when the process flow reaches this step) or manually (triggered by the user); in most use cases, this will be set to automatic
- **Required type** (options are Mandatory/Optional) - automatic actions can only be defined as mandatory. Manual actions can be defined as mandatory or optional.
- **Repeatable** - should be checked if the action can be triggered multiple times
- **Autorun Children** - when this is switched on, the child actions (the ones defined as mandatory and automatic) will run immediately after the execution of the parent action is finalized

## Back in steps

- **Allow BACK on this action** - back in process is a functionality that allows you to go back in a business process and redo a series of previous actions in the process. For more details, check [Moving a token backwards in a process](#) section.

### Action Edit

ID: 540185

Name

db77e7be-30fb-4357-8fed-23d074ede1db

Order

1

Timer Expression

Websocket Send Action

- ☒ Automatic ☐ Manual
- ☒ Mandatory ☐ Optional
- ☐ Repeatable
- Autorun Children? ☐

Save

### Parameters

The following fields are required for a minimum configuration of this type of action:

- **Message Type** - if you only want to send data, you can set this to **Default** (it defaults to the **data** message type)

#### DANGER

If you need to start a new process using a **Send data to user interface**, you can do that by setting the **Message Type** to **Action** and you will need to define a **Message** with the following format:

```
{
  "processName": "demoProcess",
  "type": "START_PROCESS_INHERIT",
  "clientDataKeys": ["webAppKeys"],
  "params": {
    "startCondition": "${startCondition}",
    "paramsToCopy": []
  }
}
```

#### ! INFO

- **paramsToCopy** - choose which of the keys from the parent process parameters to be copied to the subprocess
  - **withoutParams** - choose which of the keys from the parent process parameters are to be ignored when copying parameter values from the parent process to the subprocess
- **Message** - here you define the data to be sent as a JSON object, you can use constant values and values from the process instance data.
  - **Target Process** - is used to specify to what running process instance should this message be sent - **Active process** or **Parent process**

#### ! INFO

If you are defining this action on a **subprocess**, you can send the message to the parent process using **Target Process: Parent process**.

## Data to send

- **Keys** - are used when data is sent from the frontend via an action to validate the data (you can find more information in the [User Task configuration](#) section)

### DANGER

**Data to send** option is configurable only when the action **trigger type** is **Manual**.

#### Parameters

##### Message Type

Action

##### Message

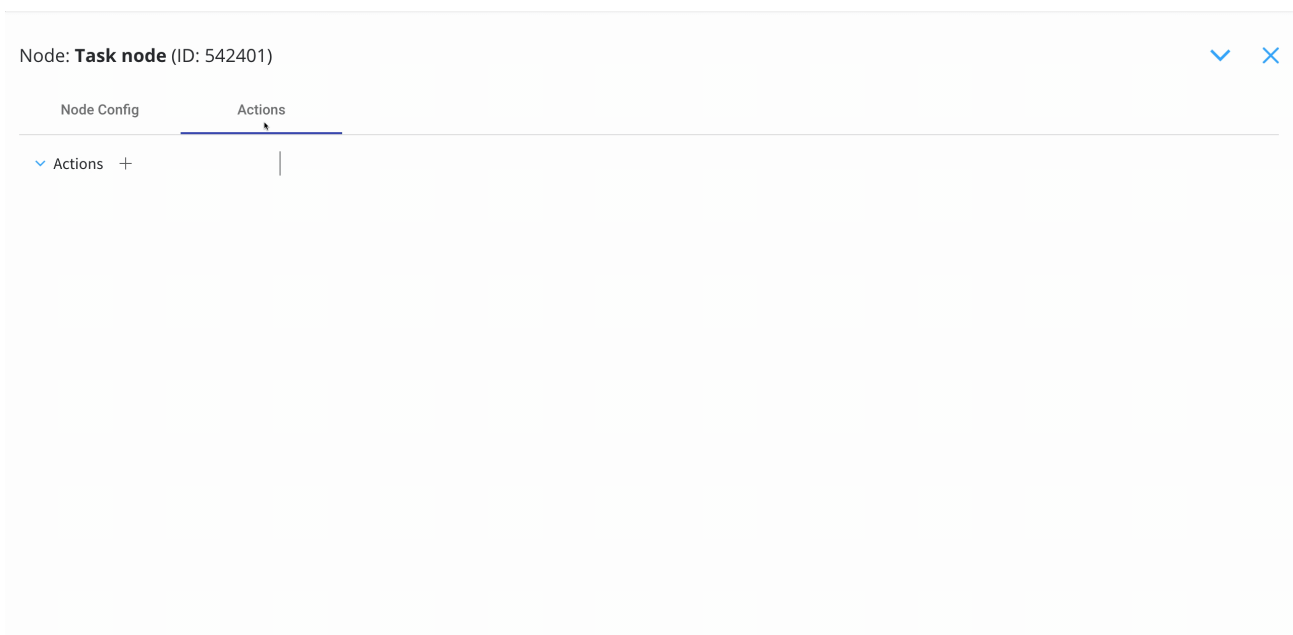
```
1 {  
2   "processName": "demoProcess",  
3   "type": "START_PROCESS_INHERIT",  
4   "clientDataKeys": ["webAppKeys"],  
5   "params": {  
6     "startCondition": "${startCondition}",  
7     "paramsToCopy": []  
8   }  
9 }
```

Save

## Send Update Data example

To send the latest value from the [process instance](#) data found at `application.client.firstName` key, to the frontend app, you can do the following:

1. Add a **Send data to user interface**.
2. Set the **Message Type** to **Default** (this is default value for `data`).
3. Add a **Message** with the data you want to send:
  - `{ "name": "${application.client.firstName}" }`
4. Choose the **Target Process**.



Was this page helpful?

## BUILDING BLOCKS / Actions / Upload File action

### ! INFO

**What is it?** An **Upload File action** is an

The fallback content to display on prerendering  
type that allows you to upload a file to a service available on  
The fallback content to display on prerendering

**Why is it useful?** The action will receive a file from the frontend and send it to Kafka, and will also attach some metadata.

## Configuring an Upload File action

Multiple options are available for this type of action and can be configured via the

The fallback content to display on prerendering

. To configure an Upload File action, use the **Actions** tab at the **task node level**, which has the following configuration options:

- **Action Edit**
- **Back in steps (for Manual actions)**
- **Parameters**
- **Data to send (for Manual actions)**

### Action Edit

- **Name** - used internally to make a distinction between different actions on nodes in the process. We recommend defining an action naming standard to be able to quickly find the process actions
- **Order** - if multiple actions are defined on the same node, the running order should be set using this option

- **Timer expression** - it can be used if a delay is required on that action. The format used for this is **ISO 8601 duration format** (for example, a delay of 30 seconds will be set up as **PT30S**)
- **Action type** - should be set to **Upload File**
- **Trigger type** (options are Automatic/Manual) - choose if this action should be triggered automatically (when the process flow reaches this step) or manually (triggered by the user); in most use cases, this will be set to automatic
- **Required type** (options are Mandatory/Optional) - automatic actions can only be defined as mandatory. Manual actions can be defined as mandatory or optional.
- **Repeatable** - should be checked if the action can be triggered multiple times
- **Autorun Children** - when this is switched on, the child actions (the ones defined as mandatory and automatic) will run immediately after the execution of the parent action is finalized

### Action Edit

ID: 540672

Name

Send Update Data

Order

1

Timer Expression

Upload File

☐ Automatic ☒ Manual

☒ Mandatory ☐ Optional

☐ Repeatable

Autorun Children? ☐

Allow BACK on this action? ☐

Save

## Back in steps

- **Allow BACK on this action** - back in process is a functionality that allows you to go back in a business process and redo a series of previous actions in the process. For more details, check [Moving a token backwards in a process](#) section.

## Parameters

- **Address** - the Kafka topic where the file will be posted
- **Document Type** - other metadata that can be set (useful for the [document plugin](#))
- **Folder** - allows you to configure a value by which the file will be identified in the future
- **Advanced configuration (Show headers)** - this represents a JSON value that will be sent on the headers of the Kafka message

## Data to send

- **Keys** - are used when data is sent from the frontend via an action to validate the data (you can find more information in the [User Task configuration](#) section)

### DANGER

**Data to send** option is configurable only when the action **trigger type** is **Manual**.

## Example



An example of **Upload File Action** is to send a file to the **document plugin**. In this case, the configuration will look like this:

### Parameters configuration

- **Address (topicName)** - will be set to (the id of the document plugin service)  
`ai.flowx.in.document.persist.v1`
- **Document Type** - metadata used by the document plugin, here we will set it to `BULK`
- **Folder** - the value by which we want to identify this file in the future (here we use the **client.id** value available on the process instance data:  
`${application.client.id}`

### Advanced configuration

- **Headers** - headers will send extra metadata to this topic -  
`{"processInstanceId": "${processInstanceId}", "destinationId": "currentNodeName"}`

Parameters

Address

ai.flowx.in.document.persist.v1

☐ Replace Values

Document Type

BULK

☐ Replace Values

Folder

\${application.client.id}

☒ Replace Values

Advanced configuration

Show Headers ☒

```
1 {"processInstanceId": ${processInstanceId}, "destinationId": "curentNodeName"}
```

☒ Replace Values

Data to send

Add Key

Save

Was this page helpful?

# BUILDING BLOCKS / Actions / Start Subprocess action

## ! INFO

**What is it?** A **Start Subprocess action** is an

The fallback content to display on prerendering that allows you to start a

The fallback content to display on prerendering from another (parent)

The fallback content to display on prerendering

.

**Why is it important?** Using **subprocesses** is a good way to split the complexity of your business flow into multiple, simple and reusable processes.

## Configuring a Start Subprocess action

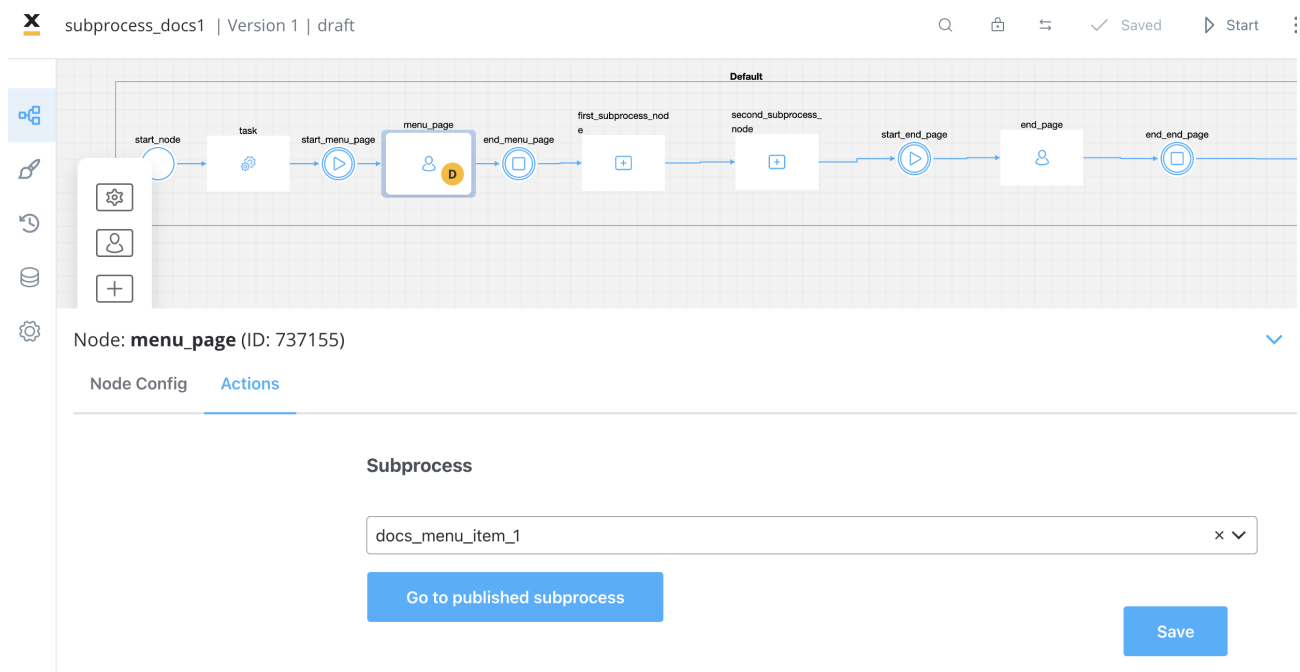
To use a process as a **subprocess**, you must first create it. Once the subprocess is created, you can start it from another (parent) process. To do this, you will need to add a **Start Subprocess** action to a **User task** node in the parent process or by using a **subprocess run node**.

Here are the steps to start a subprocess from a parent process:

1. First, create a **process** designed to be used as a **subprocess**.

2. In the parent process, create a **User task** node where you want to start the subprocess created at step 1.
3. Add a **Start Subprocess** action to the Task Node.
4. Configure the **Start Subprocess** action and from the dropdown list choose the subprocess created at step 1.

By following these steps, you can start a subprocess from a parent process and control its execution based on your specific use case.



The following properties must be configured for a **Start subprocess** action:

- **Action Edit**
- **Back in steps (for Manual actions)**
- **Parameters**
- **Data to send (for Manual actions)**

## Action Edit

- **Name** - used internally to make a distinction between different actions on nodes in the process. We recommend defining an action naming standard to be able to quickly find the process actions
- **Order** - if multiple actions are defined on the same node, the running order should be set using this option
- **Timer expression** - it can be used if a delay is required on that action. The format used for this is **ISO 8601 duration format** (for example, a delay of 30 seconds will be set up as `PT30S`)
- **Action type** - should be set to **Start Subprocess**
- **Trigger type** (options are Automatic/Manual) - choose if this action should be triggered automatically (when the process flow reaches this step) or manually (triggered by the user); in most use cases, this will be set to automatic
- **Required type** (options are Mandatory/Optional) - automatic actions can only be defined as mandatory. Manual actions can be defined as mandatory or optional.
- **Repeatable** - should be checked if the action can be triggered multiple times
- **Autorun Children** - when this is switched on, the child actions (the ones defined as mandatory and automatic) will run immediately after the execution of the parent action is finalized

## Back in steps

- **Allow BACK on this action** - back in process is a functionality that allows you to go back in a business process and redo a series of previous actions in the process. For more details, check **Moving a token backwards in a process** section.

## Parameters

- **Subprocess** - the name of the process that you want to start as a subprocess
- **Exclude from current state** - what fields do you want to exclude when copying the data from the parent process to the subprocess (by default all data fields are copied)
- **Copy from current state** - if a value is set here, it will overwrite the default behavior (of copying the whole data from the subprocess) with copying just the data that is specified (based on keys)

Node: **menu\_page** (ID: 737155)



Node Config **Actions**

### Copy from current state



Add Param

### Advanced configuration

Show Target Process ☐

Save

## Advanced configuration

- **Show Target Process** - ID of the current process, to allow the subprocess to communicate with the parent process (which is the process where this action

is configured)

## Data to send

- **Keys** - are used when data is sent from the frontend via an action to validate the data (you can find more information in the [User Task configuration](#) section)

### DANGER

**Data to send** option is configurable only when the action **trigger type** is **Manual**.

Parameters

Subprocess name

Search by name

▼

Exclude from current state

🗑

Add Param

Copy from current state

🗑

Add Param

Advanced configuration

Show Target Process ☒

`${processInstanceId}`

☒ Replace Values

Data to send

Add Key

Save

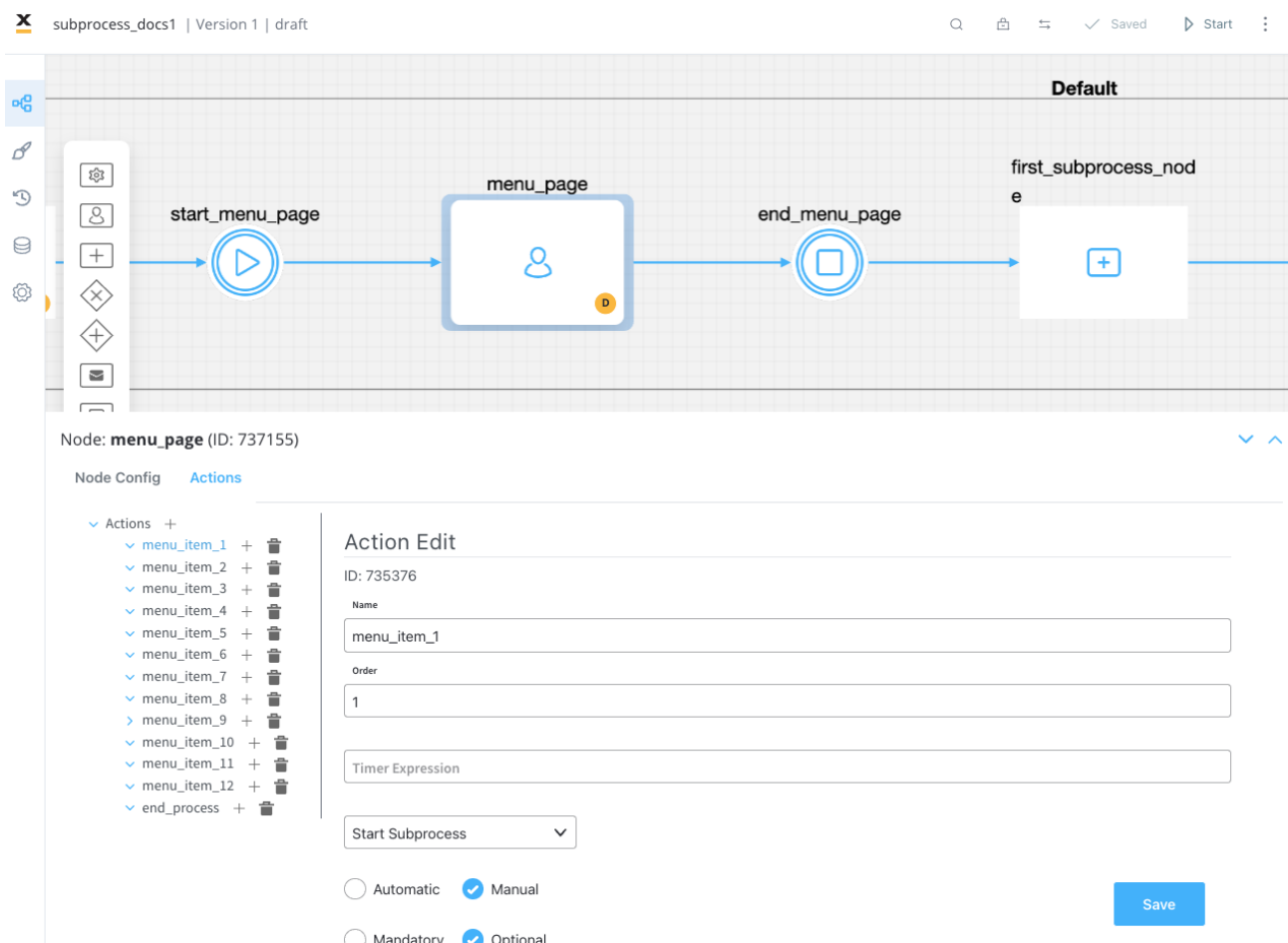
Example



Let's create a main

The fallback content to display on prerendering

, in this process we will add a user task node that will represent a menu page. In this newly added node we will add multiple subprocess actions that will represent menu items. When you select a menu item, a subprocess will run representing that particular menu item.









To start a subprocess, we can, for example, create the following minimum configuration in a user task node (now we configure the process where we want to start a subprocess):



- **Action** - `menu_item_1` - used internally to make a distinction between different actions on nodes in the process. We recommend defining an action naming standard to be able to quickly find the process actions
- **Trigger type** - Manual; Optional
- **Repeatable** - yes
- **Subprocess** - `docs_menu_item_1` - the name of the process that you want to start as a subprocess

The screenshot shows the FlowX.AI interface for configuring a subprocess action. The top bar indicates the current process is 'subprocess\_docs1' at 'Version 1' in 'draft' status. The main area is titled 'Node: menu\_page (ID: 737155)' and has two tabs: 'Node Config' and 'Actions'. The 'Actions' tab is active. Under the 'Parameters' section, the 'Subprocess' field is set to 'docs\_menu\_item\_1'. Below this is a blue button labeled 'Go to published subprocess'. The 'Exclude from current state' section contains a text input field with 'test.price' and a trash icon to its right. At the bottom of this section is a button labeled 'Add Param'.

- **Exclude from current state** - `test.price` - copy all the data from the parent, except the price data
- **Copy from current state** - leave this field empty in order to copy all the data (except the keys that are specified in the **Exclude from current state** field), if not, add the keys from which you wish to copy the data





 subprocess\_docs1 | Version 1 | draft

    Saved  Start

 Node: menu\_page (ID: 737155) 

Node Config

Actions

Copy from current state

application

application1

application2

application3

application4

application5

webSocketAddress

websocketPath

Add Param

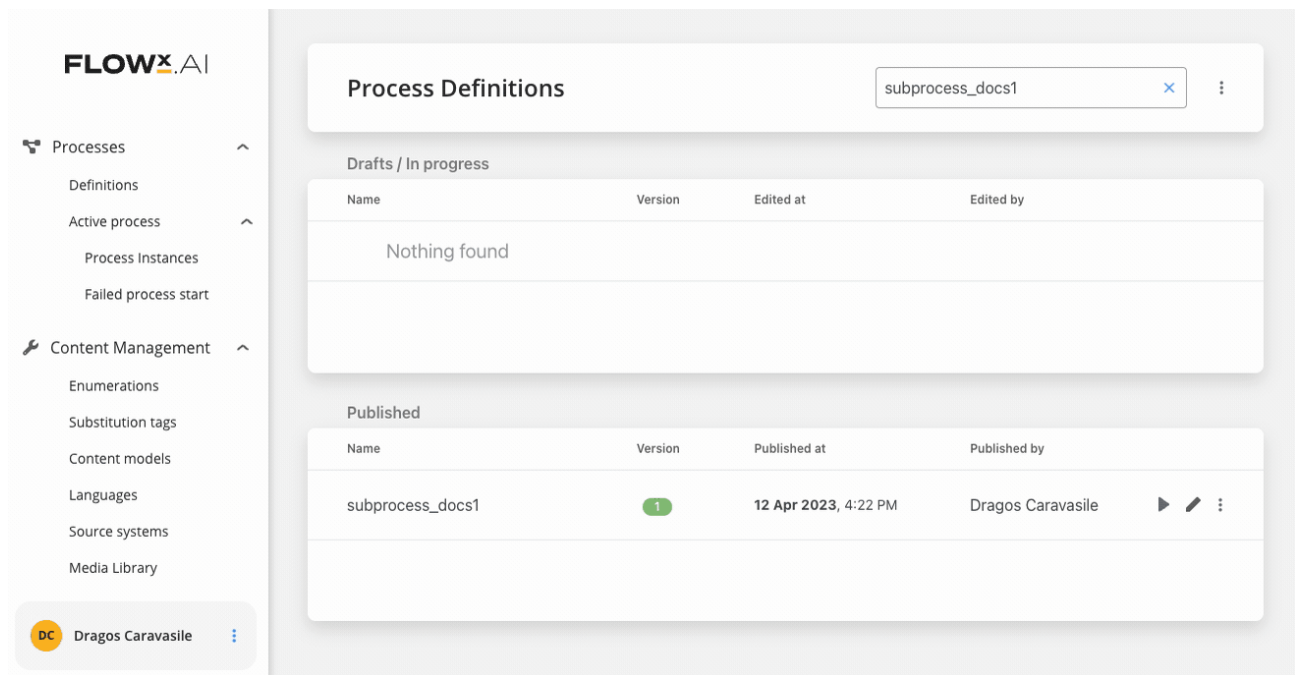
### CAUTION

When copying from the current state using a subprocess, it is mandatory to specify the `webSocketAddress` and `websocketPath` as parameters. This ensures that the Engine can accurately transmit the relevant information to the frontend, enabling it to display the appropriate UI.

## Advanced configuration

- **Target process (parentProcessInstanceId)** - `${processInstanceId}` - current process ID

## Result



**Process Definitions** subprocess\_docs1

Drafts / In progress

Name	Version	Edited at	Edited by
Nothing found			

Published

Name	Version	Published at	Published by
subprocess_docs1	1	12 Apr 2023, 4:22 PM	Dragos Caravasilie

Was this page helpful?

# BUILDING BLOCKS / Actions / Append Params to Parent Process

## ! INFO

**What is it?** It is a type of

The fallback content to display on prerendering that allows you to send data from a subprocess to a parent process.

**Why is it important?** If you are using subprocesses that produce data that needs to be sent back to the main

The fallback content to display on prerendering  
, you can do that by using an **Append Params to Parent Process** action.

## Configuring an Append Params to Parent Process

After you create a process designed to be used as a **subprocess**, you can configure the action. To do this, you need to add an **Append Params to Parent Process** on a **Task Node** in the subprocess.

The following properties must be configured:

- **Action Edit**
- **Back in steps (for Manual actions)**
- **Parameters**
- **Data to send (for Manual actions)**

### Action Edit

- **Name** - used internally to make a distinction between different actions on nodes in the process. We recommend defining an action naming standard to be able to quickly find the process actions
- **Order** - if multiple actions are defined on the same node, the running order should be set using this option
- **Timer expression** - it can be used if a delay is required on that action. The format used for this is **ISO 8601 duration format** (for example, a delay of 30 seconds will be set up as `PT30S`)

- **Action type** - should be set to **Append Params to Parent Process**
- **Trigger type** (options are Automatic/Manual) - choose if this action should be triggered automatically (when the process flow reaches this step) or manually (triggered by the user); in most use cases, this will be set to automatic
- **Required type** (options are Mandatory/Optional) - automatic actions can only be defined as mandatory. Manual actions can be defined as mandatory or optional.
- **Repeatable** - should be checked if the action can be triggered multiple times;
- **Autorun Children** - when this is switched on, the child actions (the ones defined as mandatory and automatic) will run immediately after the execution of the parent action is finalized

## Back in steps

- **Allow BACK on this action** - back in process is a functionality that allows you to go back in a business process and redo a series of previous actions in the process. For more details, check [Moving a token backwards in a process](#) section

## Parameters

- **Copy from current state** - data that you want to be copied back to the parent process
- **Destination in the parent state** - on what key to copy the param values



### TIP

To recap: if you have a **Copy from current state** with a simple **JSON** - `{"age": 17}`, that needs to be available in the parent process, on the `application.client.age` key, you will need to set this field (**Destination**

in the parent state) with `application.client`, which will be the key to append to in the parent process.

## Advanced configuration

- **Show Target Process** - ID of the parent process where you need to copy the params, this was made available on to the `${parentProcessInstanceId}` variable, if you defined it when you **started the subprocess**

## Data to send

- **Keys** - are used when data is sent from the frontend via an action to validate the data (you can find more information in the **User Task configuration** section)

### DANGER

**Data to send** option is configurable only when the action **trigger type** is **Manual**.

## Example

We have a subprocess that allows us to enter the age of the client on the **data.client.age** key, and we want to copy the value back to the parent process. The key to which we want to receive this value in the parent process is **application.client.age**.

This is the configuration to apply the above scenario:

## Parameters

- **Copy from current state** - `{"client": ${data.client.age}}` to copy the age of the client (the param value we want to copy)
- **Destination in the parent state** - `application` to append the data o to the `application` key on the parent process

### Advanced configuration

- **Show Target Process** - `${parentProcessInstanceId}` to copy the data on the parent of this subprocess



Node: **Task node** (ID: 546052)



Node Config

Actions

▼ Actions +

▼ appendToParent +

Action Edit

ID: 546651

Name

appendToParent

Order

1

Timer Expression

Append Params to Parent Process

☒ Automatic ☐ Manual

☒ Mandatory ☐ Optional

☐ Repeatable

Autorun Children? ☐

Parameters

Copy from current state

```
1 {"client":${data.client.age}}
```

☒ Replace Values

Destination in the parent state

application

☐ Replace Values

Advanced configuration

Save

Was this page helpful?