FLOWX.AI

BUILDING BLOCKS / Node

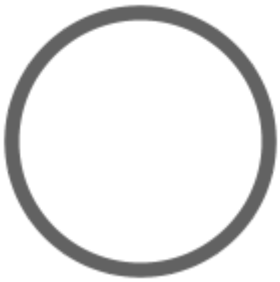
Contents

BUILDING BLOCKS / Node / Start/End nodes

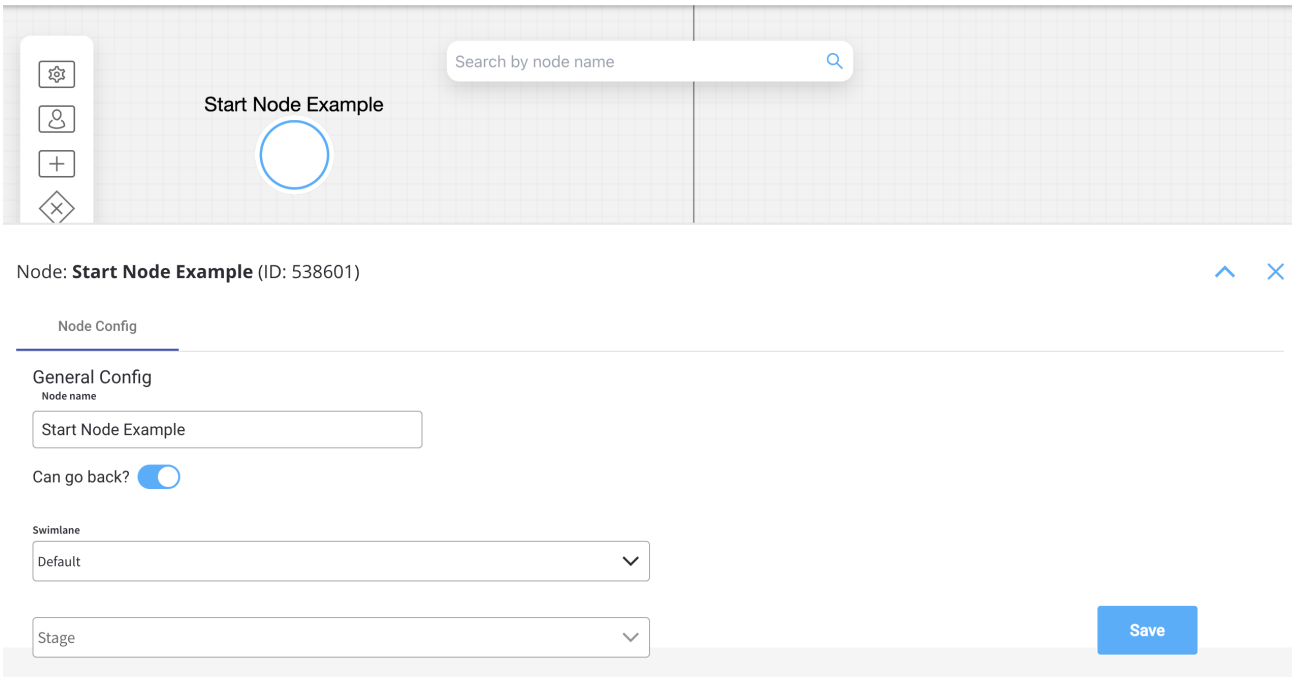
Let's go through all the options for configuring start and end nodes for a process definition.

Start node

The start node represents the beginning of a process and it is mandatory to add one when creating a process.



A process can have one or more start nodes. If you defined multiple start nodes, each should have a start condition value configured. When starting a new process instance the desired start condition should be used.



Configuring a start node

Node configuration is done by accessing the **Node Config** tab. You have the following configuration options for a **start node**:

- General Config
- Start condition

General Config

- **Node name** - the name of the node
- **Can go back** - switching this option to true will allow users to return to this step after completing it

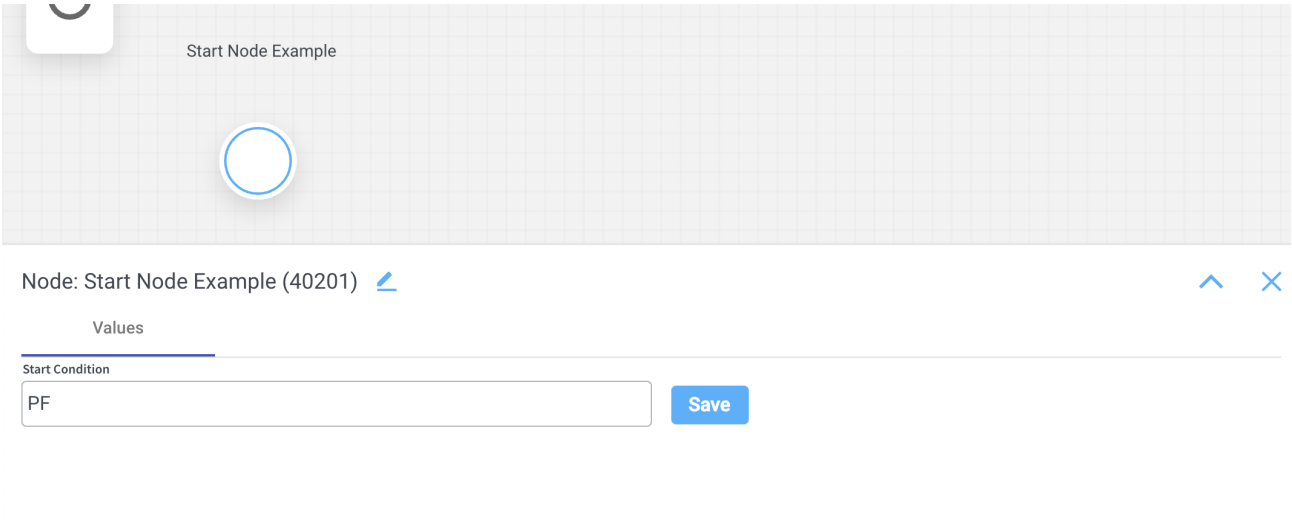
> **INFO**

> When encountering a step with `canGoBack` switched to false, all steps found behind it will become unavailable.

- **Swimlane** - choose a swimlane (if there are multiple swimlanes on the process) to make sure only certain user roles have access only for certain process nodes- if there are no multiple swimlanes, the value is **Default**
- **Stage** - assign a stage to the node

Start condition

The start condition should be set as a string value. This string value will need to be set on the payload for the start process request on the `startCondition` key.



To test the start condition, we can send a start request via REST:

```
POST {{processUrl}}/api/process/{{processName}}/start
{
    "startCondition": "PF"
}
```

Error handling on start condition

If a request is made to start a process with a start condition that does not match any start node, an error will be generated. Let's take the previous example and assume we send an incorrect value for the start condition:

```
POST {{processUrl}}/api/process/{{processName}}/start
{
    "startCondition": "PJ"
}
```

A response with the error code `bad request` and title `Start node for process definition not found` will be sent in this case:

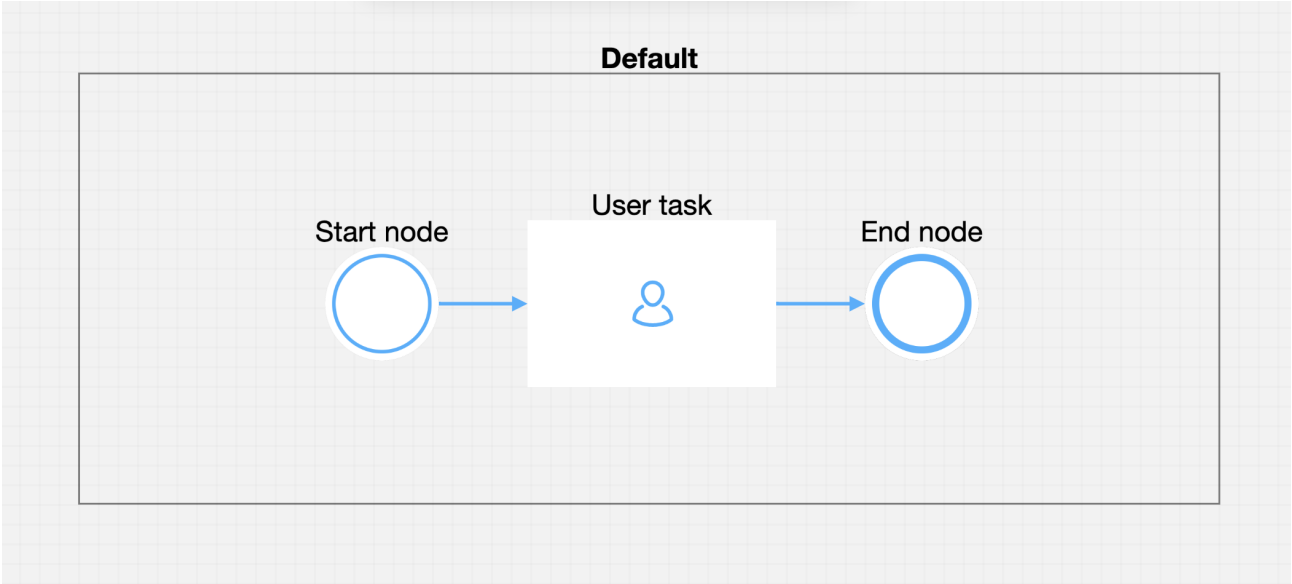
```
{
    "entityName": "ai.flowx.process.definition.domain.NodeDefi
    "defaultMessage": "Start node for process definition not f
    "errorKey":
"error.validation.process_instance.start_node_for_process_def_
    "type": "https://www.jhipster.tech/problem/problem-with-me
    "title": "Start node for process definition not found.",
    "status": 400,
    "message":
"error.validation.process_instance.start_node_for_process_def_
    "params": "ai.flowx.process.definition.domain.NodeDefiniti
}
```

End node

An end node is used to mark where the process finishes. When the process reaches this node, the process is considered completed and its status will be set to `Finished`.

Configuring an end node

Multiple end nodes can be used to show different end states. The configuration is similar to the start node.



Was this page helpful?

BUILDING BLOCKS / Node / Message send/Message received task nodes

Message send task and **message received**

The fallback content to display on prerendering
are used to handle the interaction between a running process and any external systems.

Message send task

This node is used to configure messages that should be sent to external systems.

Configuring a message send task node

Node configuration is done by accessing the **Node Config** tab. You have the following configuration options for a message send task node:

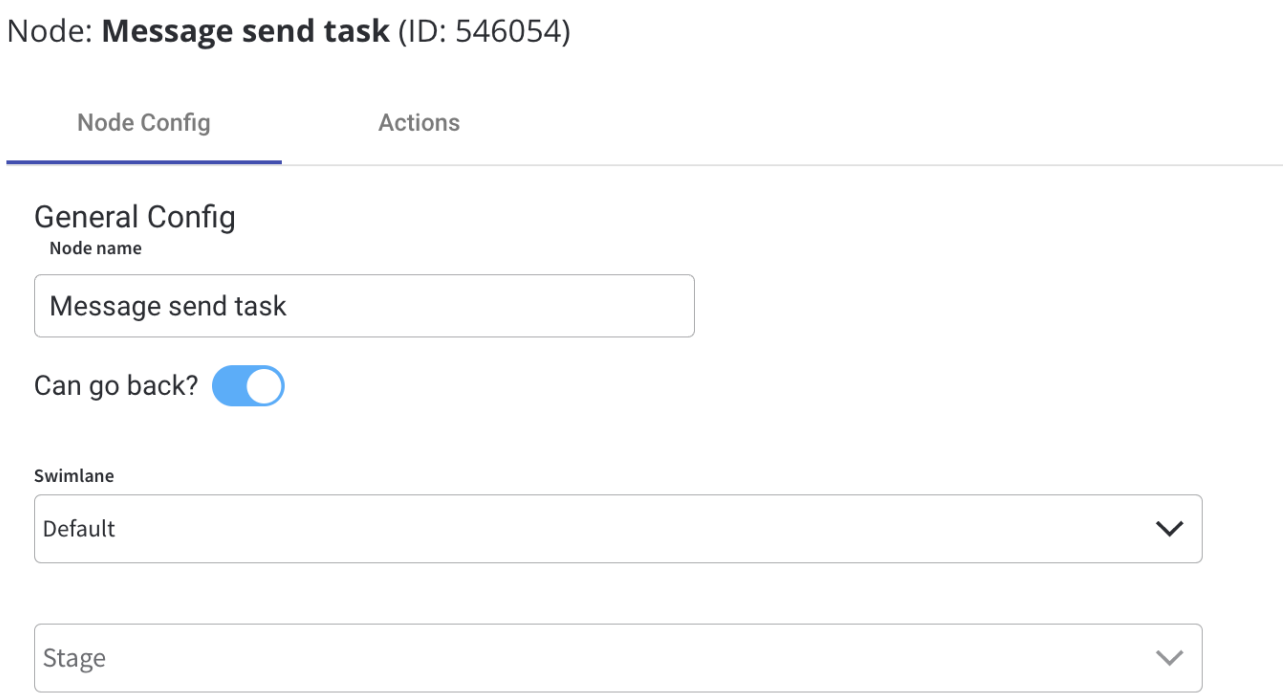
General Config

Inside the General Config you have the following properties:

- **Node name** - the name of the node
- **Can Go Back** - switching this option to true will allow users to return to this step after completing it

> **INFO**
>
> When encountering a step with `canGoBack` switched to false, all steps found behind it will become unavailable.

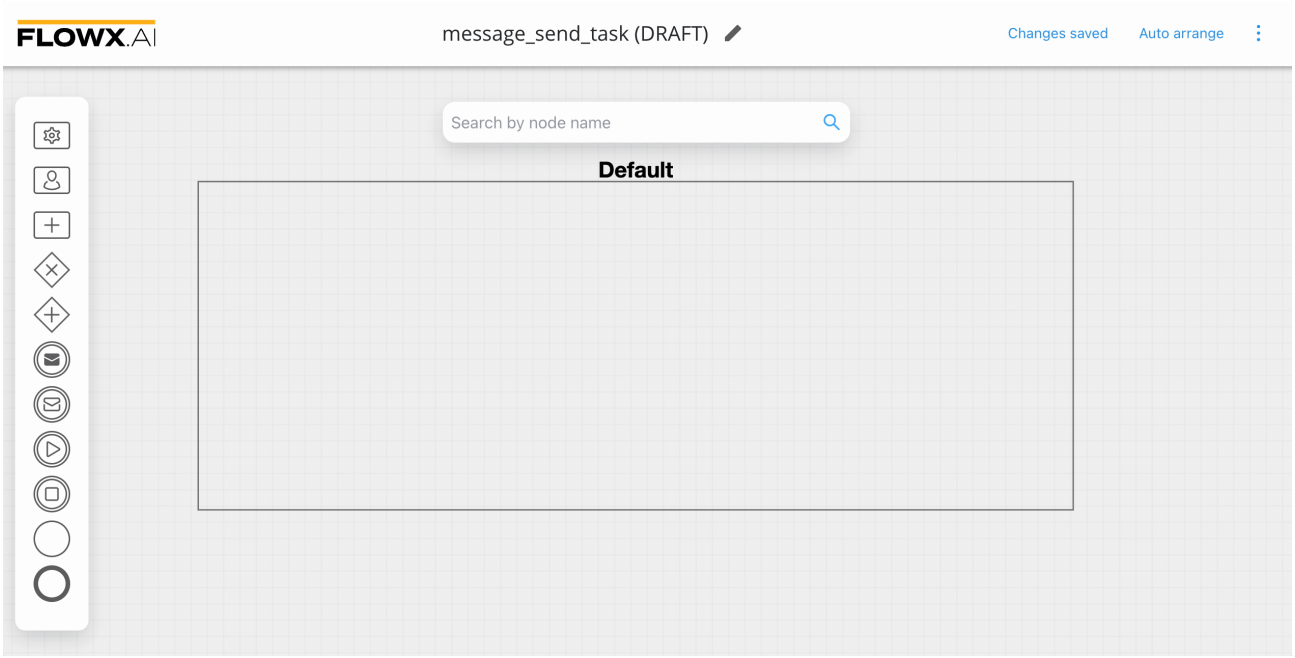
- **Swimlane** - choose a swimlane (if there are multiple swimlanes on the process) to make sure only certain user roles have access only for certain process nodes - if there are no multiple swimlanes, the value is **Default**
- **Stage** - assign a stage to the node



To configure a message send task node, we first need to add a new node and then configure an

The fallback content to display on prerendering
(**Kafka Send Action** type):

1. Open
   The fallback content to display on prerendering
   and start configuring a process.
2. Add a **message send task** node.
3. Select the **message send task** node and open **node configuration**.
4. Add an
   The fallback content to display on prerendering
   , the type of the action set to **Kafka send**.
5. ❗ A few action parameters will need to be filled in depending on the selected action type.



Multiple options are available for this type of action and can be configured via the FLOWX.AI Designer. To configure and add an action to a node, use the

The fallback content to display on prerendering
tab at the 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 easily 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 **Kafka Send Action** for actions used to send messages to external systems
- **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

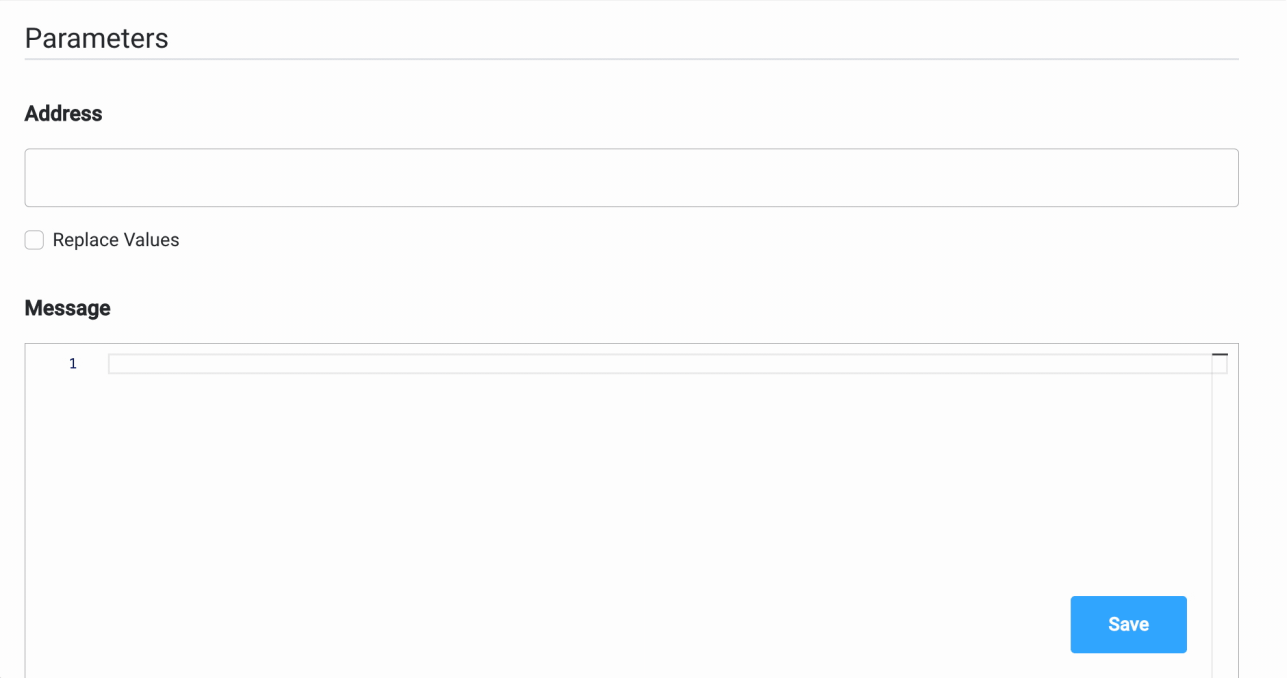
- **Allow BACK on this action** - back in process is a functionality that allows you to go back in a business process and redo a series of previous actions in the process, or more details, check **Moving a token backwards in a process** section

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**.



For more information about what Kafka is, check the following sections:

» Intro to Kafka

» Kafka documentation

Example of a message send event

Send a message to a CRM integration to request a search in the local database:

Action Edit

- **Name** - pick a name that makes it easy to figure out what this action does, for example, `sendRequestToSearchClient`
- **Order** - 1
- **Timer Expression** - this remains empty if we want to action to be triggered as soon as the token reaches this node
- **Action type** - Kafka Send Action
- **Trigger type** - *Automatic* - to trigger this action automatically
- **Required type** - *Mandatory* - to make sure this action will be run before advancing to the next node
- **Repeatable** - false, it only needs to run once

Parameters

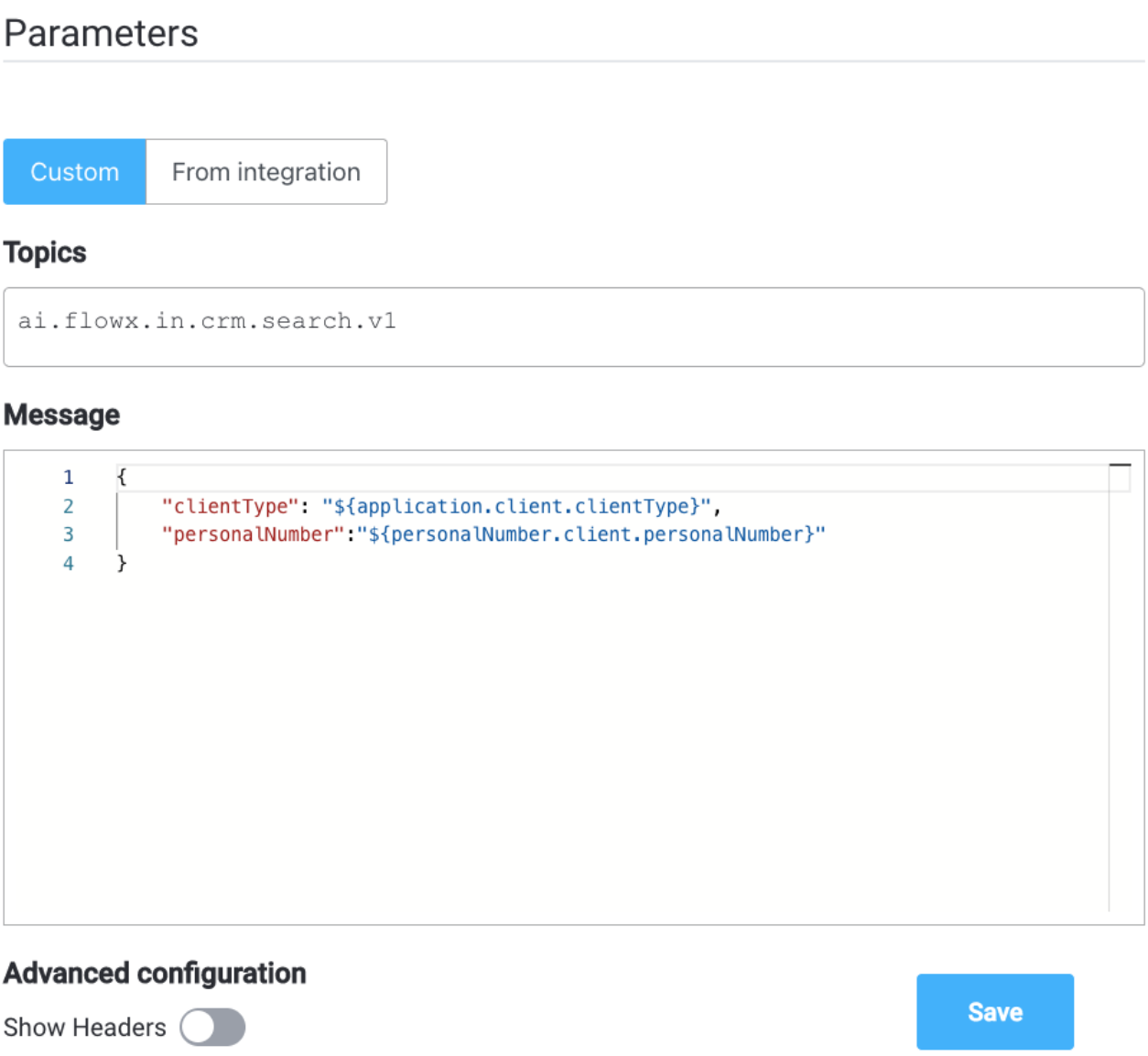
> **INFO**
>
> Parameters can be added either using **Custom** option (where you configure everything on the spot), or by using **From integration** and import parameters already defined in an integration.
>
> More details about **Integrations management** you can find here.

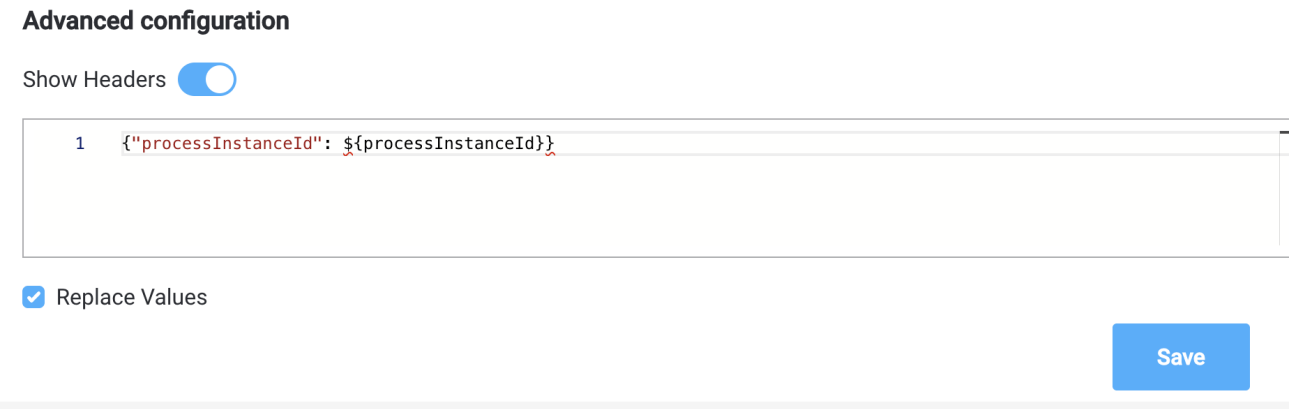
Custom

- **Topics** - `ai.flowx.in.crm.search.v1` the Kafka topic on which the CRM listens for requests
- **Message** - `{ "clientType": "${application.client.clientType}", "personalNumber": "${personalNumber.client.personalNumber}"`

`}` - the message payload will have two keys, `clientType` and `personalNumber`, both with values from the process instance

- **Headers** - `{"processInstanceId": ${processInstanceId}}`


Parameters
Custom
From integration
Topics
ai.flowx.in.crm.search.v1
Message
{
"clientType": "${application.client.clientType}",
"personalNumber":"${personalNumber.client.personalNumber}"
}
Advanced configuration
Show Headers
Save

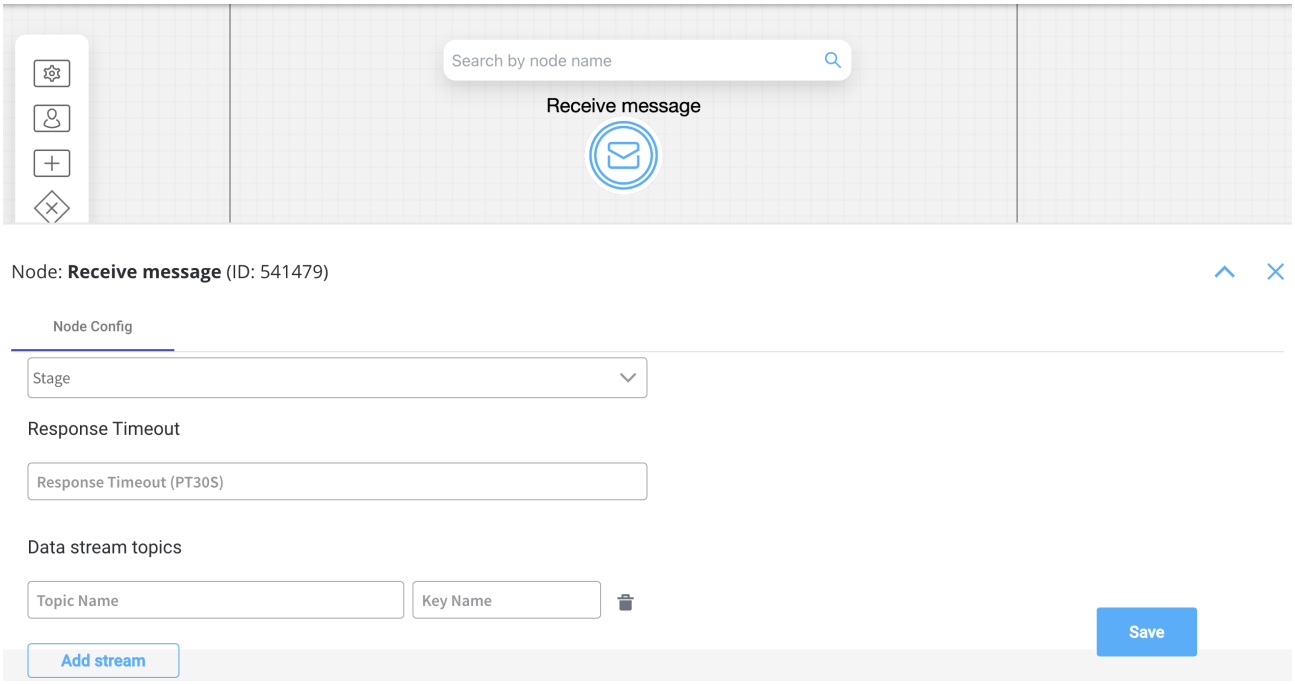


Message receive task

This type of node is used when we need to wait for a reply from an external system.

The reply from the external system will be saved in the process instance values, on a specified key. If the message needs to be processed at a later time, a timeout can be set using the ISO 8601 format.

For example, let's think about a CRM microservice that waits to receive requests to look for a user in a database. It will send back the response when a topic is configured to listen for the response.



Configuring a message receive task node

The values you need to configure for this node are the following:

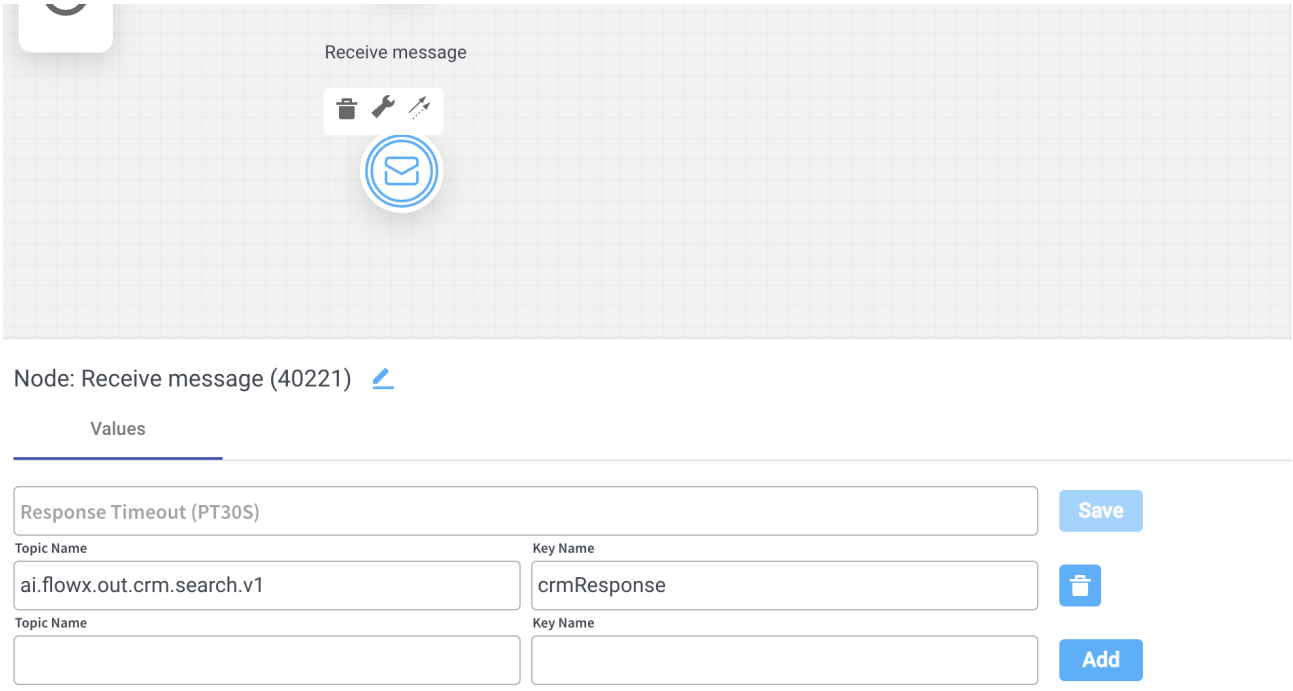
- **Topic name** - the topic name where the process engine listens for the response (this should be added to the platform and match the topic naming rule for the engine to listen to it) - `ai.flowx.out.crm.search.v1`

DANGER

A naming pattern must be defined on the process engine to use the defined topics. It is important to know that all the events that start with a configured pattern will be consumed by the Engine. For example, `KAFKA_TOPIC_PATTERN` is the topic name pattern that the Engine listens to for incoming Kafka events.

- **Key Name** - will hold the result received from the external system, if the key already exists in the process values, it will be overwritten - `crmResponse`

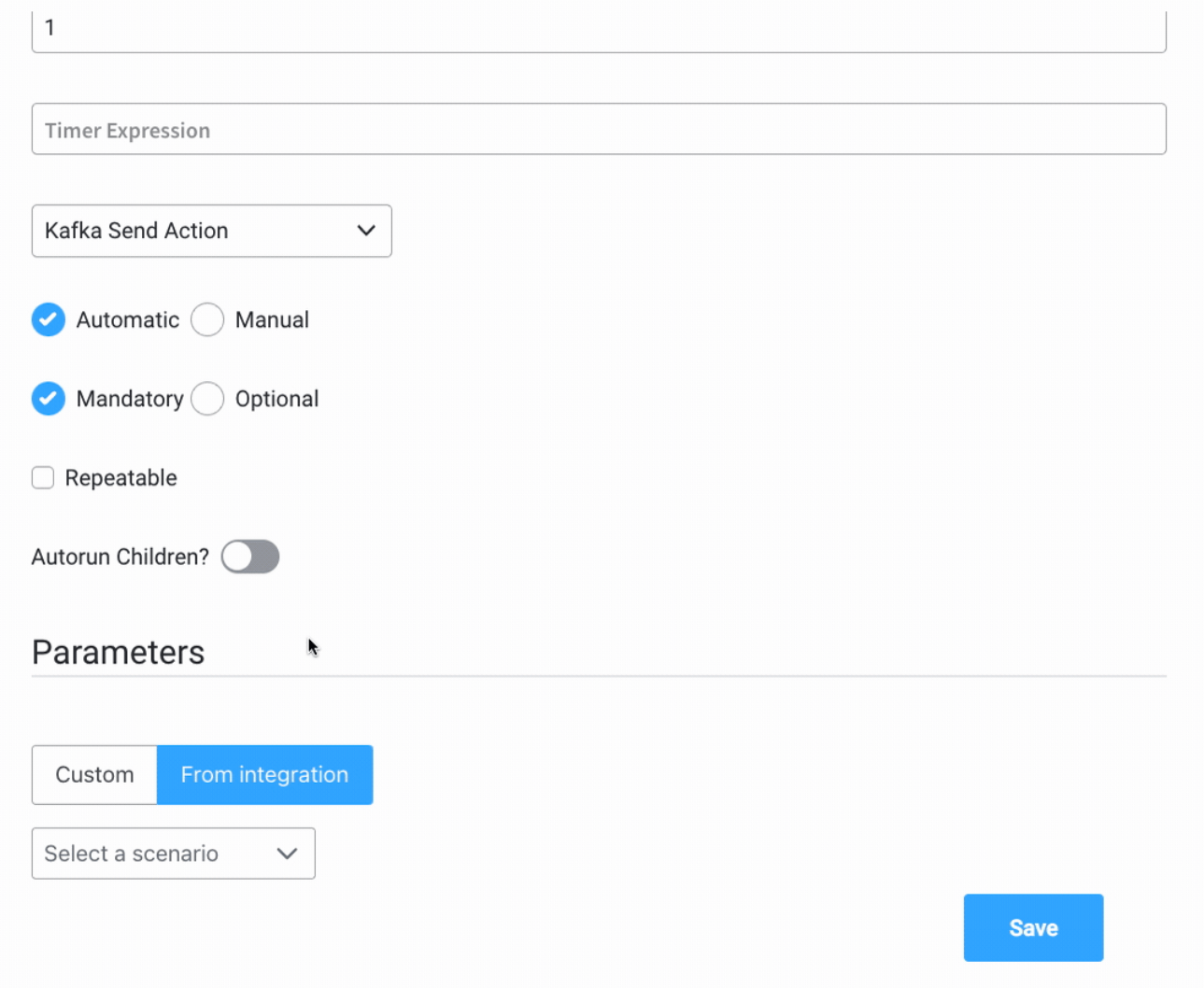
For more information about Kafka configuration, click here.



From integration

After defining one integration (inside Integration management) you can open a compatible node and start using already defined integrations.

- **Topics** - topics defined in your integration
- **Message** - the **Message data model** from your integration
- **Headers** - all integrations have `processInstanceId` as a default header parameter, add any other relevant parameters



Was this page helpful?

BUILDING BLOCKS / Node / Task node

A task

The fallback content to display on prerendering
refers to a task that utilizes various services, such as Web services, automated applications, or other similar services, to accomplish a particular task.

This type of node finds application in multiple scenarios, including:

- Executing a
  The fallback content to display on prerendering
  on the process instance data.
- Initiating a
  The fallback content to display on prerendering
  .
- Transferring data from a
  The fallback content to display on prerendering
  to the parent process.
- Transmitting data to
  The fallback content to display on prerendering
  .

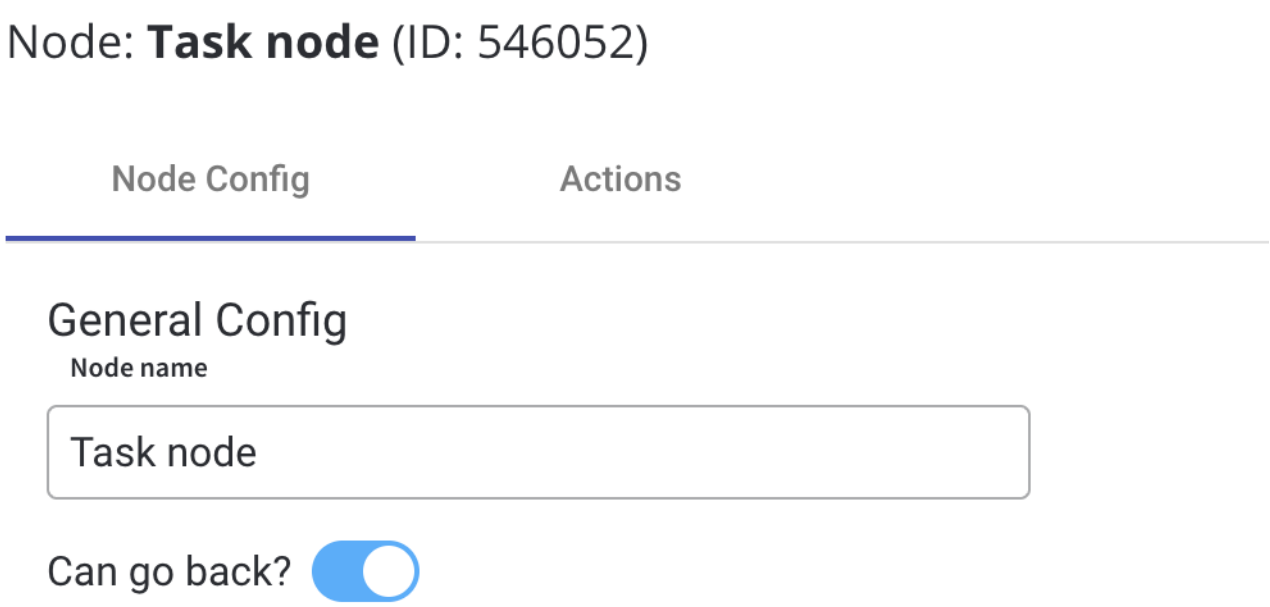
Configuring task nodes

One or more actions can be configured on a task node. The actions are executed in the configured order.

Node configuration is done by accessing the **Node Config** tab. You have the following configuration options for a task node:

General Config

- **Node name** - the name of the node
- **Can go back** - switching this option to true will allow users to return to this step after completing it



> **INFO**
>
> When encountering a step with `canGoBack` switched to false, all steps found behind it will become unavailable.

- **Swimlane** - choose a swimlane (if there are multiple swimlanes on the process) to make sure only certain user roles have access only for certain

process nodes- if there are no multiple swimlanes, the value is **Default**

- **Stage** - assign a stage to the node

Response Timeout

- **Response timeout** - can be triggered if, for example, a topic that you define and add in the Data stream topics tab does not respect the pattern, the format used for this is ISO 8601 duration format(for example, a delay of 30s will be set up like `PT30S`)

Data stream topics

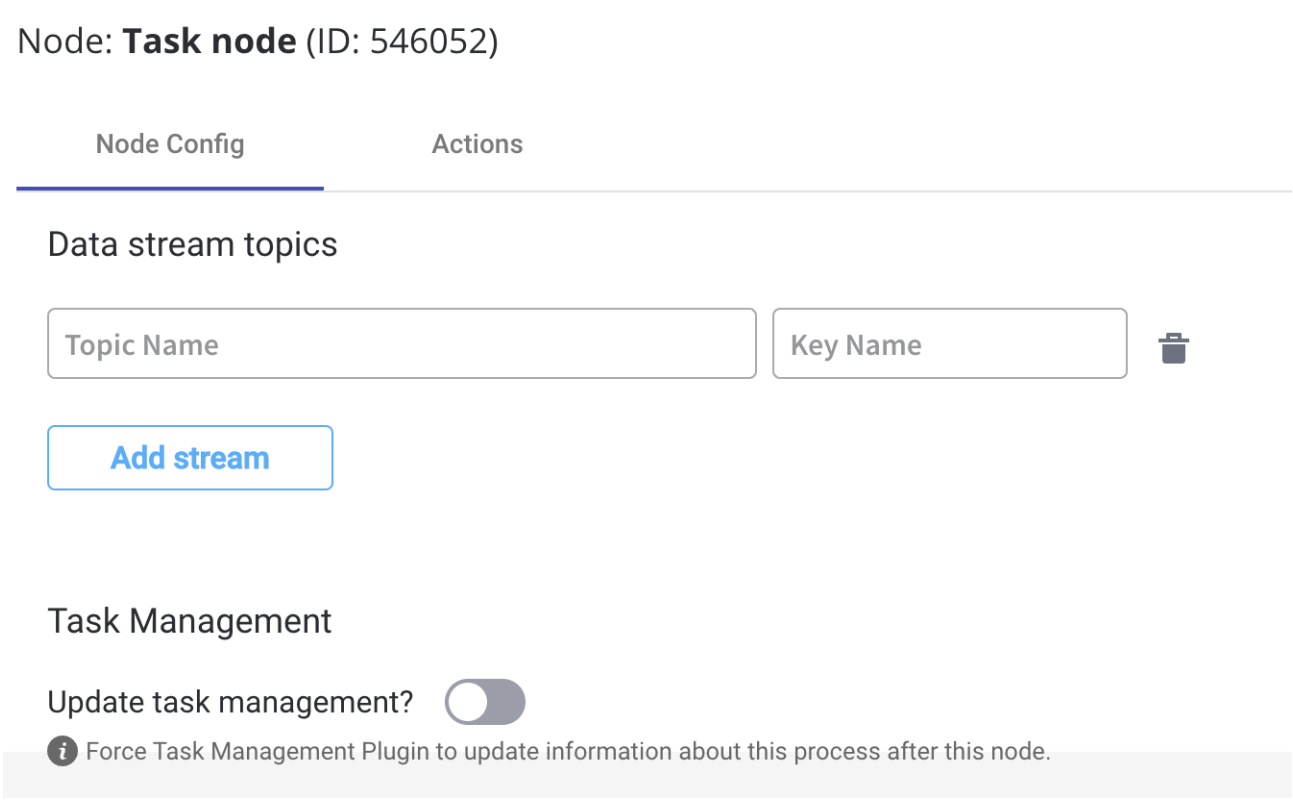
- **Topic Name** - the topic name where the process engine listens for the response (this should be added to the platform and match the topic naming rule for the engine to listen to it) - available for UPDATES topics (Kafka receive events)

> **DANGER**
>
> A naming pattern must be defined on the process engine configuration to use the defined topics. It is important to know that all the events that start with a configured pattern will be consumed by the Engine. For example, `KAFKA_TOPIC_PATTERN` is the topic name pattern where the Engine listens for incoming Kafka events.

- **Key Name** - will hold the result received from the external system, if the key already exists in the process values, it will be overwritten

Task Management

- **Update task management** - force Task Manager Plugin to update information about this process after this node



Configuring task nodes actions

Multiple options are available when configuring an action on a task node. To configure and add an action to a node, use the **Actions** tab at the node level, which has the following configuration options:

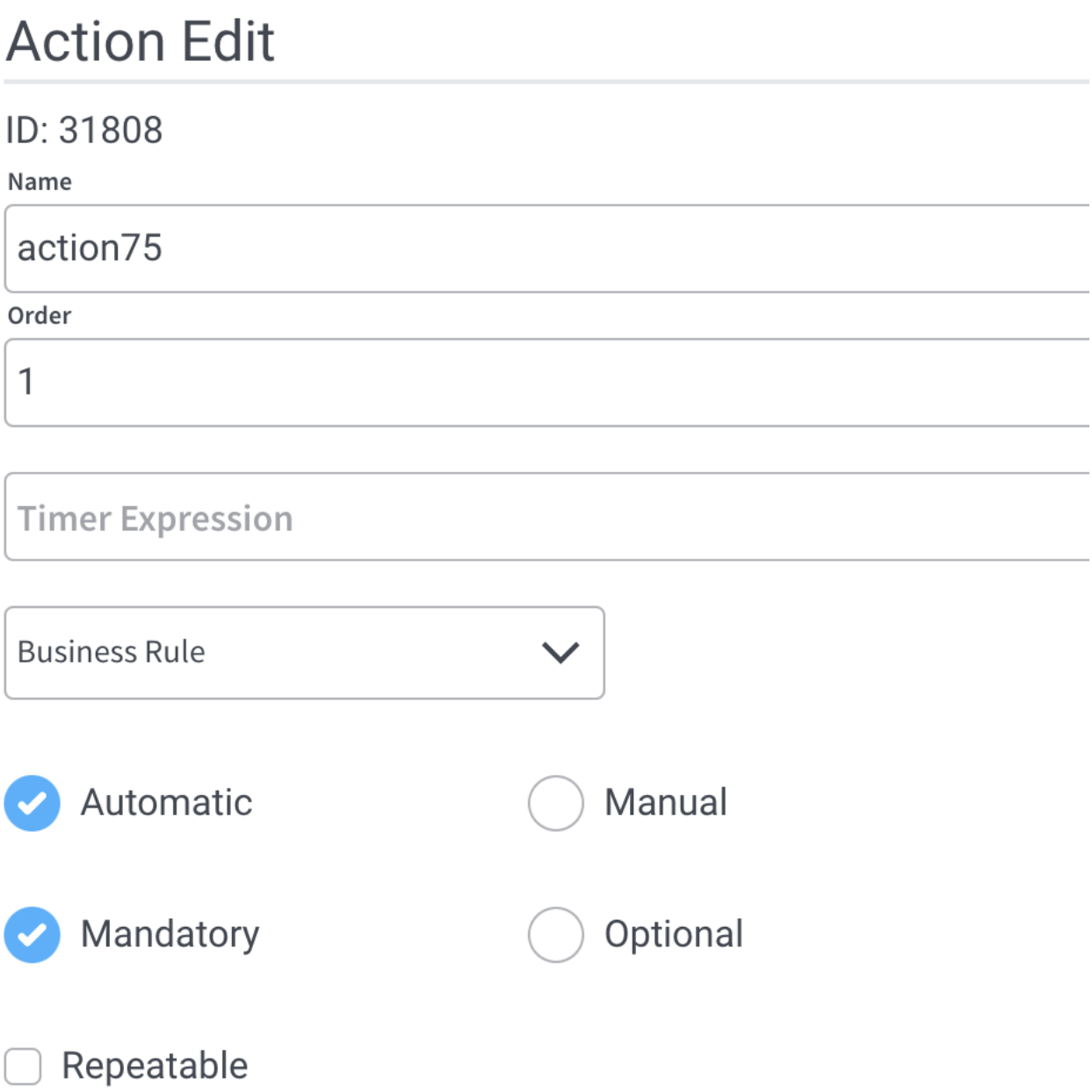
- Action Edit
- Parameters

Action Edit

> **INFO**

> Depending on the type of the **action**, different properties are available, let's take a **Business rule** as an example.

1. **Name** - used internally to differentiate between different actions on nodes in the process. We recommend defining an action naming standard to be able to quickly find the process actions.
2. **Order** - if multiple actions are defined on the same node, their running order should be set using this option
3. **Timer Expression** - 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 30s will be set up like `PT30S`)
4. **Action type** - defines the appropriate action type
5. **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.
6. **Required type** - (options are Mandatory/Optional) - automatic actions can only be defined as mandatory. Manual actions can be defined as mandatory or optional.
7. **Repeatable** - should be checked if the action can be triggered multiple times



Parameters

INFO

> Depending on the type of the **action**, different properties are available. We refer to a **Business rule** as an example

1. **Business Rules** - business rules can be attached to a node by using actions with action rules on them, these can be specified using DMN rules, MVEL expressions, or scripts written in Javascript, Python, or Groovy.

» Supported scripting languages

Business Rule action

A business rule is a Task action that allows a script to run. For now, the following script languages are supported:

- MVEL
- JavaScript
- Python
- Groovy
- DMN - more details about a DMN business rule configuration can be found here

For more details on how to configure a Business Rule action, check the following section:

» Business rule action

Send data to user interface

Being an event-driven platform FLOWX uses web socket communication in order to push events from the frontend application. For more details on how to configure a Send data to user interface action, check the following section:

» Send data to user interface

Upload File action

Upload file action will be used to upload a file from the frontend application and send it via a Kafka topic to the document management system.

For more details on how to configure an Upload File action, check the following section:

» Upload file action

Start Subprocess action

In order to create reusability between business processes, as well as split complex processes into smaller, easier-to-maintain flows, the start subprocess business rule can be used to trigger the same sequence multiple times.

For more details on how to configure a Business Rule action, check the following section:

» Start subprocess action

Append Params to Parent Process

Used for copying data in the subprocess from its parent process. For more details about the configuration, check the following section:

» Append params to parent process

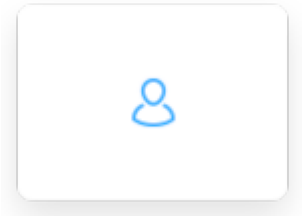
Was this page helpful?

BUILDING BLOCKS / Node / User task node

This

The fallback content to display on prerendering
represents an interaction with the user. It is used to display a piece of UI (defined in the UI Designer) or a custom Angular component. You can also define
The fallback content to display on prerendering
available for the users to interact with the process.

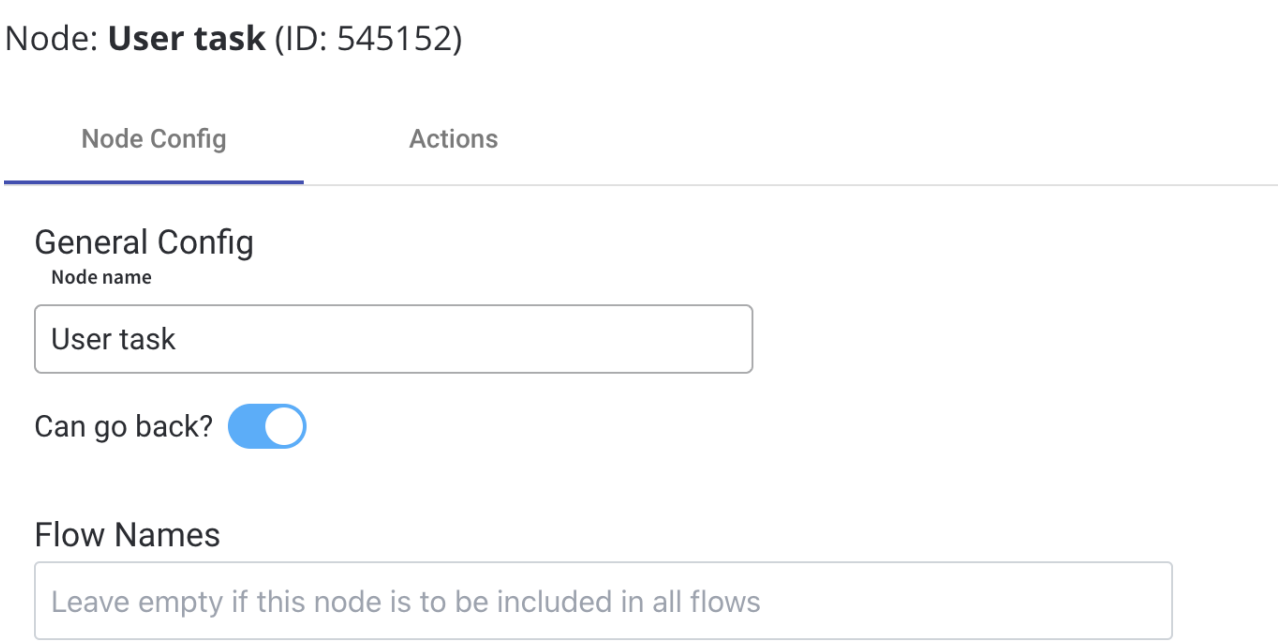
Configuring a user task node



User task nodes allow you to define and configure UI templates and possible actions for a certain template config node (ex: button components).

General Config

- **Node name** - the name of the node
- **Can go back** - setting this to true will allow users to return to this step after completing it. When encountering a step with `canGoBack` false, all steps found behind it will become unavailable.
- **Flow Names** - leave this field empty if the node should be included in all flows

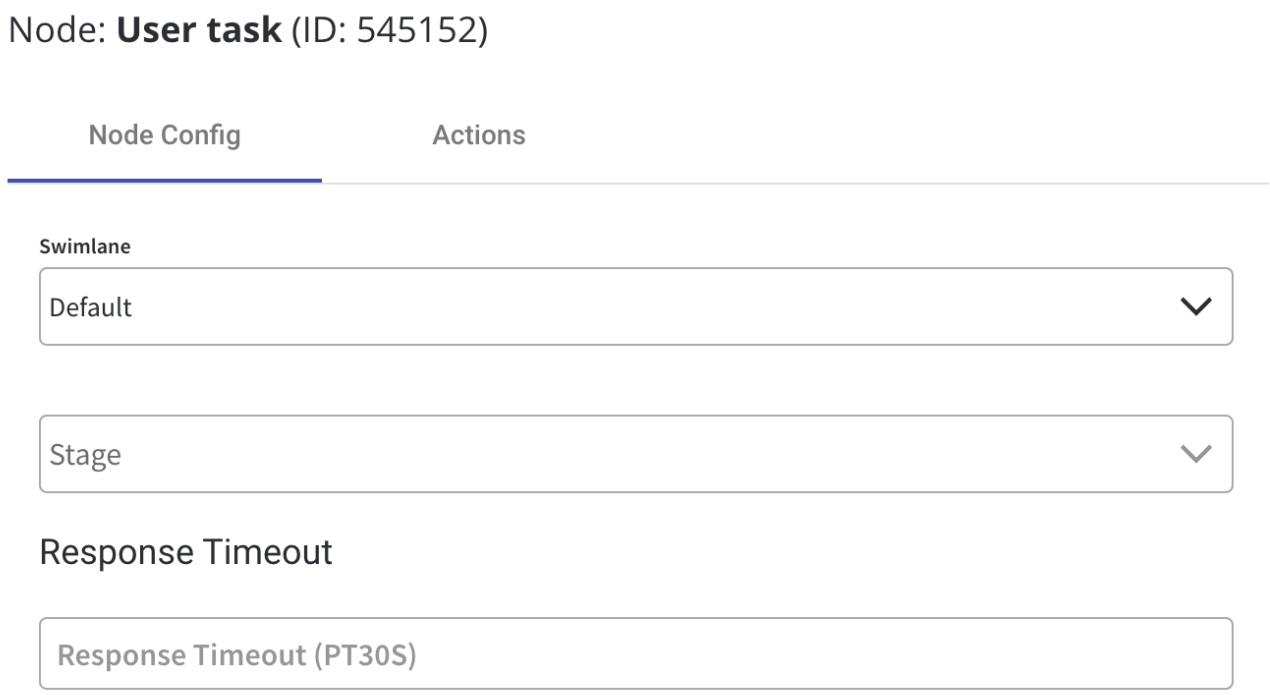


> **INFO**
>
> When encountering a step with `canGoBack` switched to false, all steps found behind it will become unavailable.

- **Swimlane** - choose a swimlane (if there are multiple swimlanes on the process) to make sure only certain user roles have access only for certain process nodes- if there are no multiple swimlanes, the value is **Default**
- **Stage** - assign a stage to the node

Response Timeout

- **Response timeout** - can be triggered if, for example, a topic that you define and add in the Data stream topics tab does not respect the pattern, the format used for this is ISO 8601 duration format (for example, a delay of 30s will be set up like `PT30S`)



The fallback content to display on prerendering

- **Topic Name** - the topic name where the
  The fallback content to display on prerendering
  listens for the response (this should be added to the platform and match the topic naming rule for the engine to listen to it) - available for UPDATES topics (Kafka receive events)

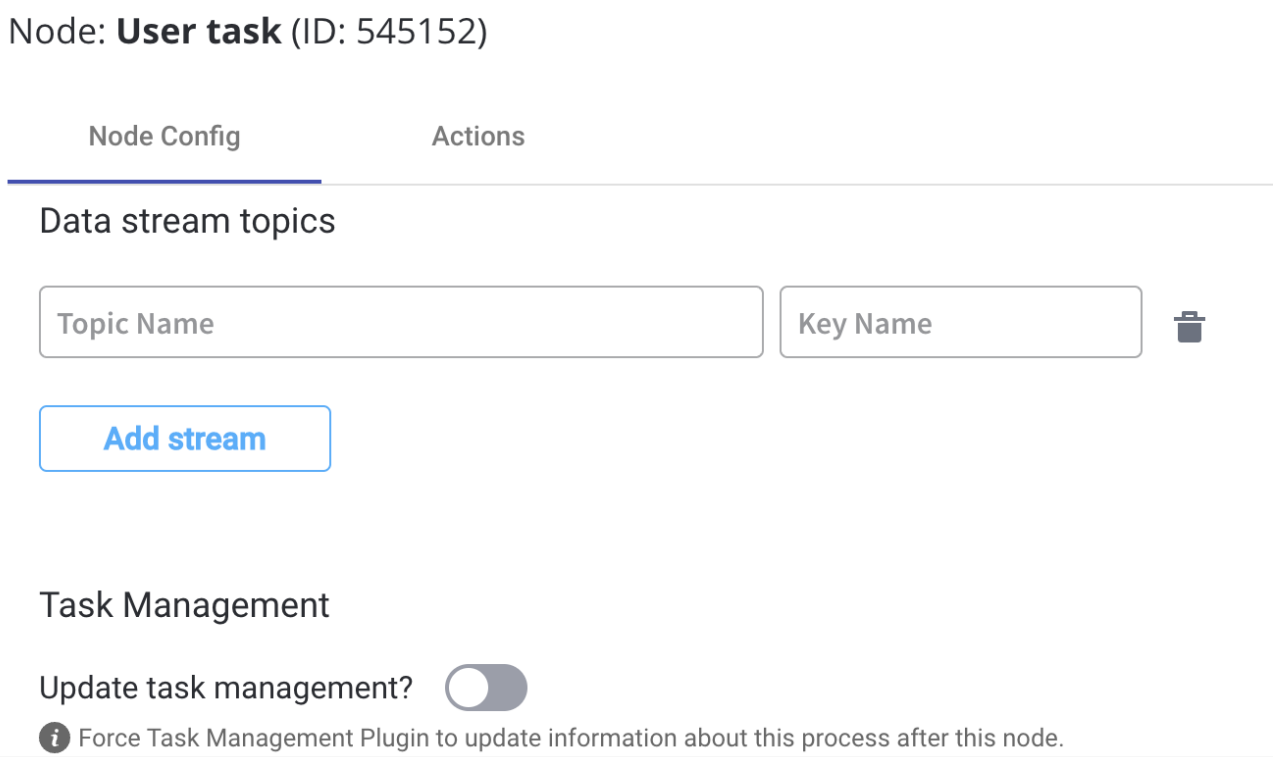
DANGER

A naming pattern must be defined on the process engine configuration to use the defined topics. It is important to know that all the events that start with a configured pattern will be consumed by the Engine. For example, `KAFKA_TOPIC_PATTERN` is the topic name pattern where the Engine listens for incoming Kafka events.

- **Key Name** - will hold the result received from the external system, if the key already exists in the process values, it will be overwritten

Task Management

- **Update task management** - force Task Management
  The fallback content to display on prerendering
  to update information about this process after this node



Configuring the UI

The

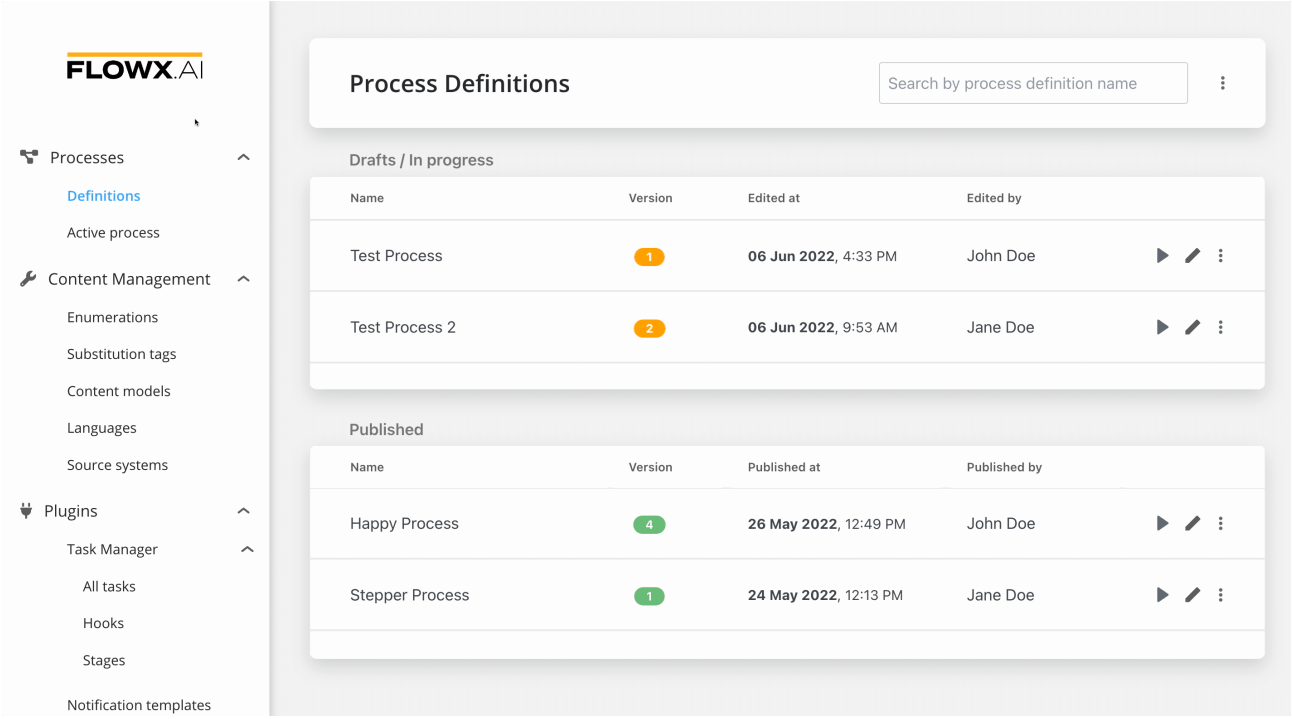
The fallback content to display on prerendering
includes an intuitive UI Designer (drag-and-drop editor) for creating diverse UI templates. You can use various elements from basic buttons, indicators, and forms, but also predefined collections or prototypes.

Accessing the UI Designer

To access the

The fallback content to display on prerendering
, follow the next steps:

1. Open **FLOWX Designer** and from the **Processes** tab select **Definitions**.
2. Select a **process** from the process definitions list.
3. Click the **Edit process** button.
4. Select a **user task node** from the Pro dcess Designer then click the **brush** icon to open the **UI Designer**.



» Creating a user interface

Predefined components

UI can be defined using the available components provided by FLOWX, using the UI Designer available at node level.

Predefined components can be split in 3 categories:

1. Root components

These elements are used to group different types of components, each having a different purpose:

- **Card** - used to group and configure the layout for multiple **form elements.**

- **Container** - used to group and configure the layout for multiple **components** of any type.
- **Custom** - these are Angular components developed in the container application and passed to the SDK at runtime, identified here by the component name

More details in the following section:

» Root components

2. UI Components

The root component can hold a hierarchical component structure.

Available children for **Card** and **Container** are:

- **Container** - used to group and align its children
- **Form** - used to group and align form field elements (**inputs**, **radios**, **checkboxes**, etc)
- **Image** - allows you to configure an image in the document
- **Text** - a simple text can be configured via this component, basic configuration is available
- **Hint** - multiple types of hints can be configured via this component
- **Link** - used to configure a hyperlink that opens in a new tab
- **Button** - Multiple options are available for configuration, the most important part being the possibility to add actions
- **File Upload** - A specific type of button that allows you to select a file

- **Custom** - custom components

More details in the following section:

» Component types

3. Form elements

This type of elements are used to allow the user to input data, and can be added only in a **Form** Component. They have have multiple properties that can be managed.

1. **Input** - FLOWX form element that allows you to generate an input form filed
2. **Select** - to add a dropdown
3. **Checkbox** - the user can select zero or more input from a set of options
4. **Radio** - the user is required to select one and only one input from a set of options
5. **Datepicker** - to select a date from a calendar picker
6. **Switch** - allows the user to toggle an option on or off

More details in the following section:

» Form elements

Custom components

These are components developed in the web application and referenced here by component identifier. This will dictate where the component is displayed in the component hierarchy and what actions are available for the component.

To add a custom component in the template config tree, we need to know its unique identifier and the data it should receive from the process model.

More details in the following section:

» Custom

The sections that can be configured are as follows:

1. **Message** - configure what data will be pushed to the frontend application
2. **Input keys** - used to define the process model paths from which the components will receive its data
3. **UI Actions** - actions defined here will be made available to the custom component. Multiple actions can be configured on a custom component and mapped to different triggers when developing it. Naming each action suggestively is important so the frontend engineer developing the component knows what actions should be triggered by certain events.

More information about configuration, [here](using ui designer).

Displaying a UI element

When a process instance is started the web application will receive all the UI elements that can be displayed in that process.

When the process instance token will reach a User Task, a web socket message will be sent informing the SDK to display the UI element associated with that user task

Example:

1. Start a process: **POST**
   `{{processUrl}}/api/internal/process/DemoProcess/start`

> **INFO**
>
> The provided instruction involves initiating a process by making a `POST` request to the specified URL (`{{processUrl}}/api/internal/process/DemoProcess/start`). This API call triggers the start of a process named "DemoProcess" by sending relevant data to the server.


```
{
"processDefinitionName" : "DemoProcess",
  "tokens" : [ {
    "id" : 759224,
    "startNodeId" : null,
    "currentNodeId" : 662807,
    "currentNodeName" : null,
    "state" : "ACTIVE",
    "statusCurrentNode" : "ARRIVED",
    "dateUpdated" : "2023-05-31T09:44:39.969634Z",
    "uuid" : "d310996d-f3b9-44e5-983d-3631c844409e"
  } ],
  "state" : "STARTED",
  "templateConfig" : [ {
    "id" : 630831,
```

```
    "flowxUuid" : "80ea0a85-2b0b-442a-a123-2480c7aa2dce",
    "nodeDefinitionId" : 662856,
    "componentIdentifier" : "CONTAINER",
    "type" : "FLOWX",
    "order" : 1,
    "canGoBack" : true,
    "displayOptions" : {
      "flowxProps" : { },
      "style" : null,
      "flexLayout" : {
        "fxLayoutGap" : 0,
        "fxLayoutAlign" : "start stretch",
        "fxLayout" : "column"
      },
      "className" : null,
      "platform" : "DEFAULT"
    },
    "templateConfig" : [ {
      "id" : 630832,
      "flowxUuid" : "38e2c164-f8cd-4f6e-93c8-39b7cdd734cf",
      "nodeDefinitionId" : 662856,
      "uiTemplateParentId" : 630831,
      "componentIdentifier" : "TEXT",
      "type" : "FLOWX",
      "order" : 0,
      "key" : "",
      "canGoBack" : true,
      "displayOptions" : {
        "flowxProps" : {
          "text" : "Demo text"
        },
        "style" : null,
        "flexLayout" : null,
        "className" : null,
        "platform" : "DEFAULT"
```

```
      },
      "expressions" : {
        "hide" : ""
      },
      "templateConfig" : [ ],
      "dataSource" : {
        "processData" : {
          "parentFlowxUuid" : null
        },
        "nomenclator" : {
          "parentFlowxUuid" : null
        }
      }
    } ]
  } ],
  "uuid" : "44177340-5ac6-4591-89ad-04df0815fbd0",
  "generalData" : null,
  "backCounter" : 0,
  "startedByActionId" : null,
  "subProcesses" : null,
  "subprocessesUuids" : null,
  "baseUrl" : null
}
```

2. **ProgressUpdateDto** will trigger the **SDK** to search for the UI element having the same **nodeId** with the one in the SSE event.

3. Additionally, it will ask for data and actions that are required for this component via a **GET request** `{{processUrl}}/api/process/db573705-71dd-4216-9d94-5ba2fb36ff2a/data/42062`

```
...
nodeDefinitionId" : 662856,
      "processDefinitionId" : 662952,
      "actionParams" : [ {
        "id" : 759458,
        "key" : "headers",
        "value" : "{\"processInstanceId\":
${processInstanceId}}",
        "replaceValues" : true,
        "actionDefinitionId" : 759403
      }, {
        "id" : 759457,
        "key" : "customId",
        "value" : "folder",
        "replaceValues" : true,
        "actionDefinitionId" : 759403
      }, {
        "id" : 759456,
        "key" : "documentType",
        "value" : "document",
        "replaceValues" : false,
        "actionDefinitionId" : 759403
      }, {
        "id" : 759455,
        "key" : "topicName",
        "value" : "test.topic",
        "replaceValues" : false,
        "actionDefinitionId" : 759403
      } ],
      "actionRuleDefinitions" : [ ],
      "callbackActions" : null,
      "timerExpression" : "",
      "order" : 1,
      "manual" : false,
```

```
        "repeatable" : false,
        "optional" : false,
        "autoRunChildren" : false,
        "allowTokenReset" : false,
        "restartFromSnapshot" : false,
        "keysForRestart" : [ ],
        "keys" : [ ]
  ...
```

Values

For more details, please check the following page:

» Message send receive task

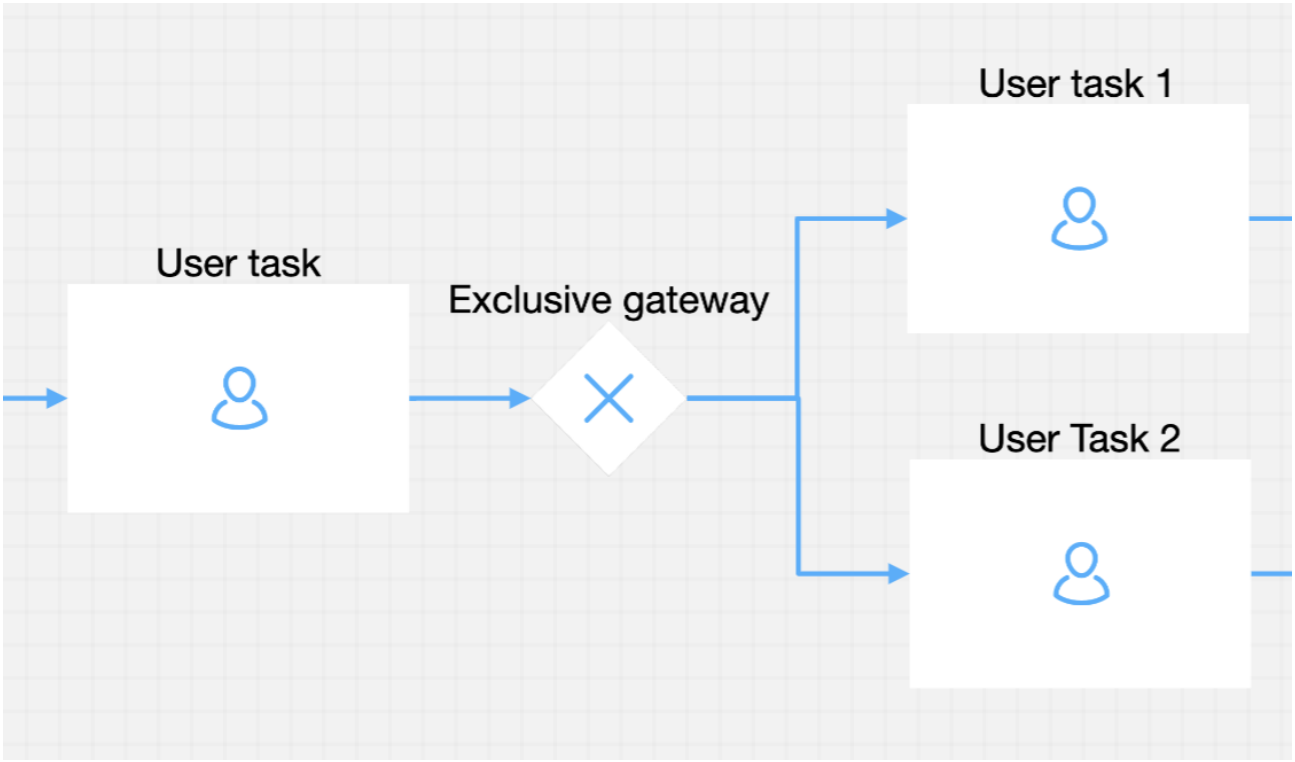
Was this page helpful?

BUILDING BLOCKS / Node / Exclusive gateway

The fallback content to display on prerendering
decisions can be configured using an Exclusive Gateway. Using this
The fallback content to display on prerendering
will make `if condition then go to this node` constructions are available.

Configuring an Exclusive gateway node

To configure this kind of node, it is useful to previously configure the **in** and **out** sequence from the gateway process.

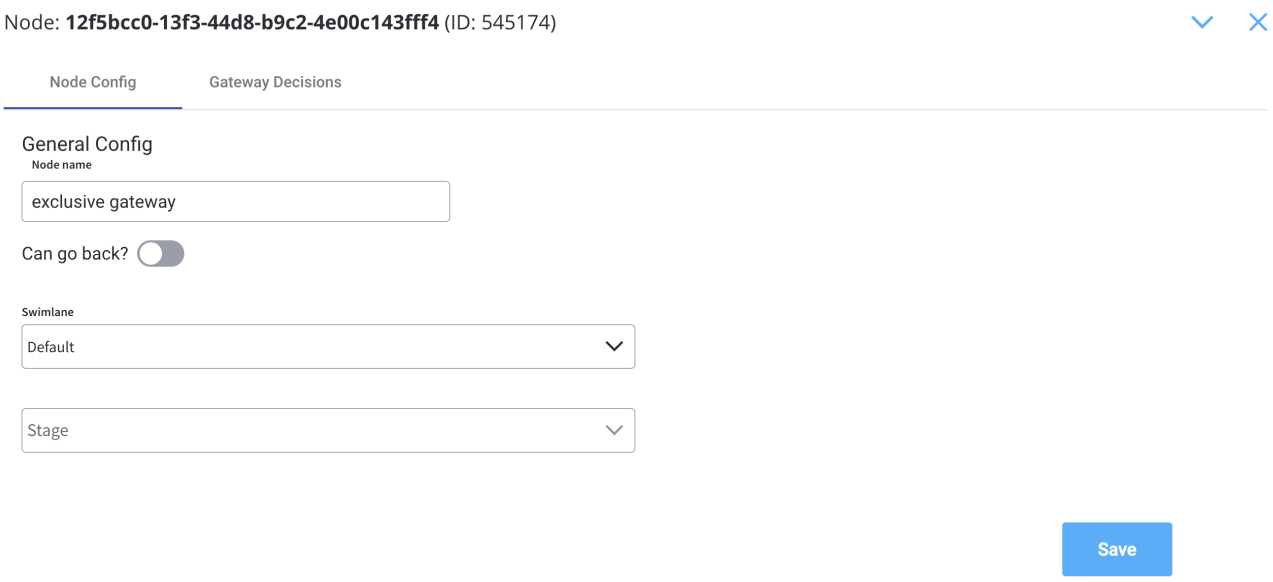


General Config

- **Node name** - the name of the node
- **Can go back** - setting this to true will allow users to return to this step after completing it

> **INFO**
>
> When encountering a step with `canGoBack` switched to false, all steps found behind it will become unavailable.

- **Swimlane** - choose a swimlane (if there are multiple swimlanes on the process) to make sure only certain user roles have access only for certain process nodes- if there are no multiple swimlanes, the value is **Default**
- **Stage** - assign a stage to the node

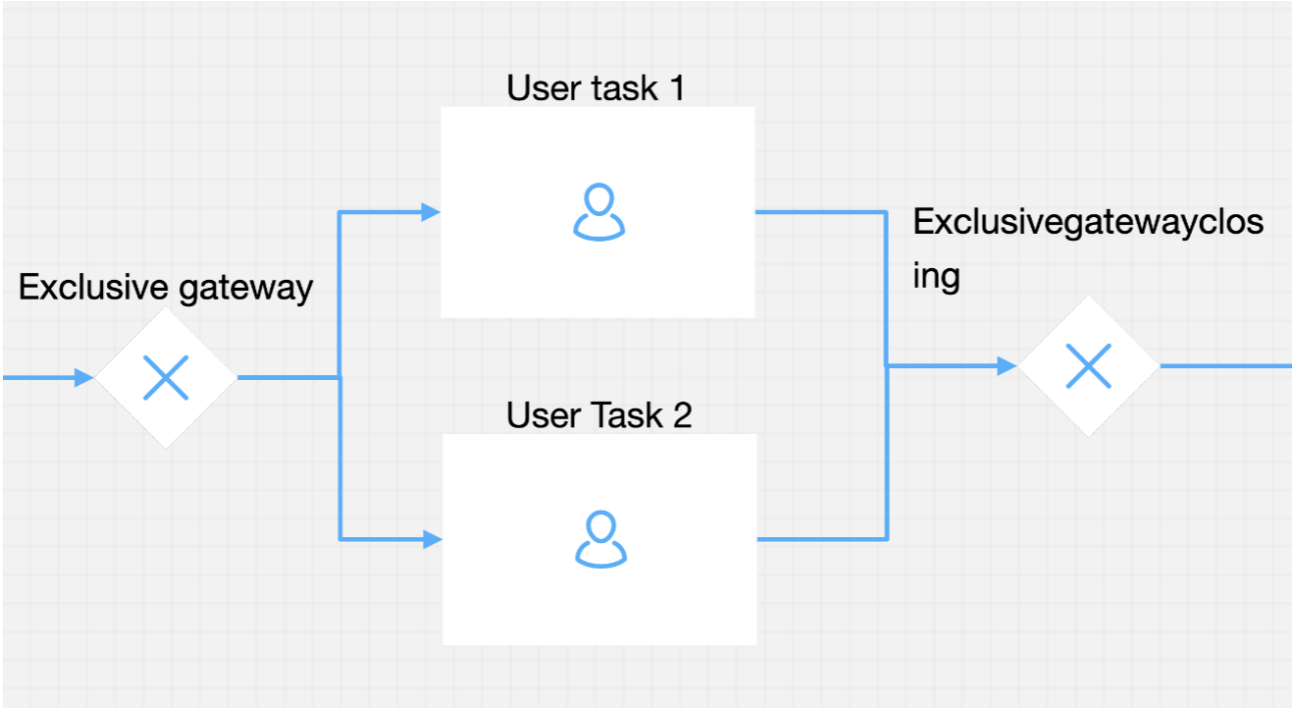


Gateway Decisions

- **Language** - when configuring the condition, MVEL (or DMN) will be used and you should enter an expression that will be evaluated as **true** or **false**
- **Conditions** - selecting the **Gateway Decisions** tab of the gateway we can see that we can configure a list of conditions (**if, else if, else**) and **select** from a dropdown where we should go if the condition is **true**

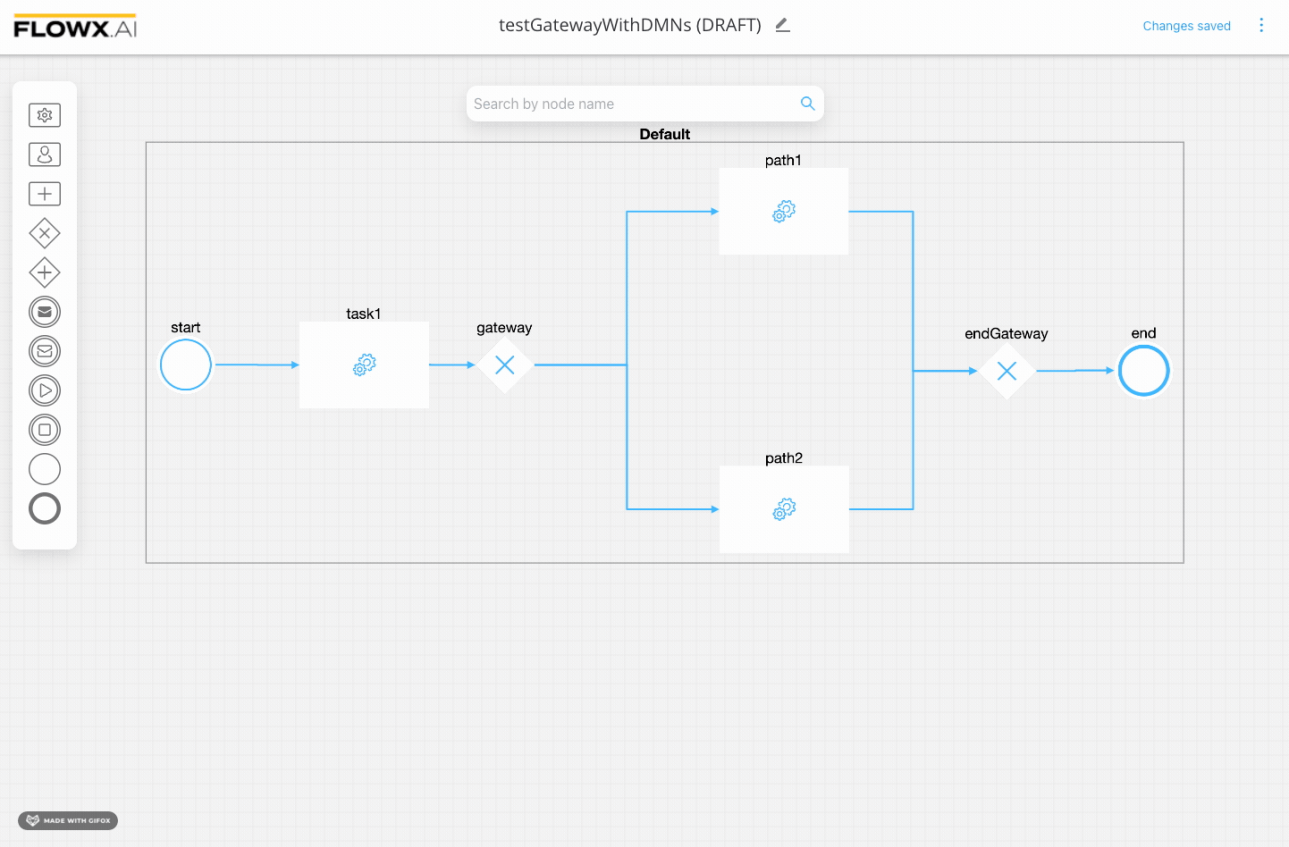
> **DANGER**
>
> Expression order is important because the first **true** evaluation will stop the execution and the token will move to the selected node.

After the exclusive part of the process (where a path or another will be used), you need to end each path or join back to a single process using a new exclusive gateway without any configuration on it.

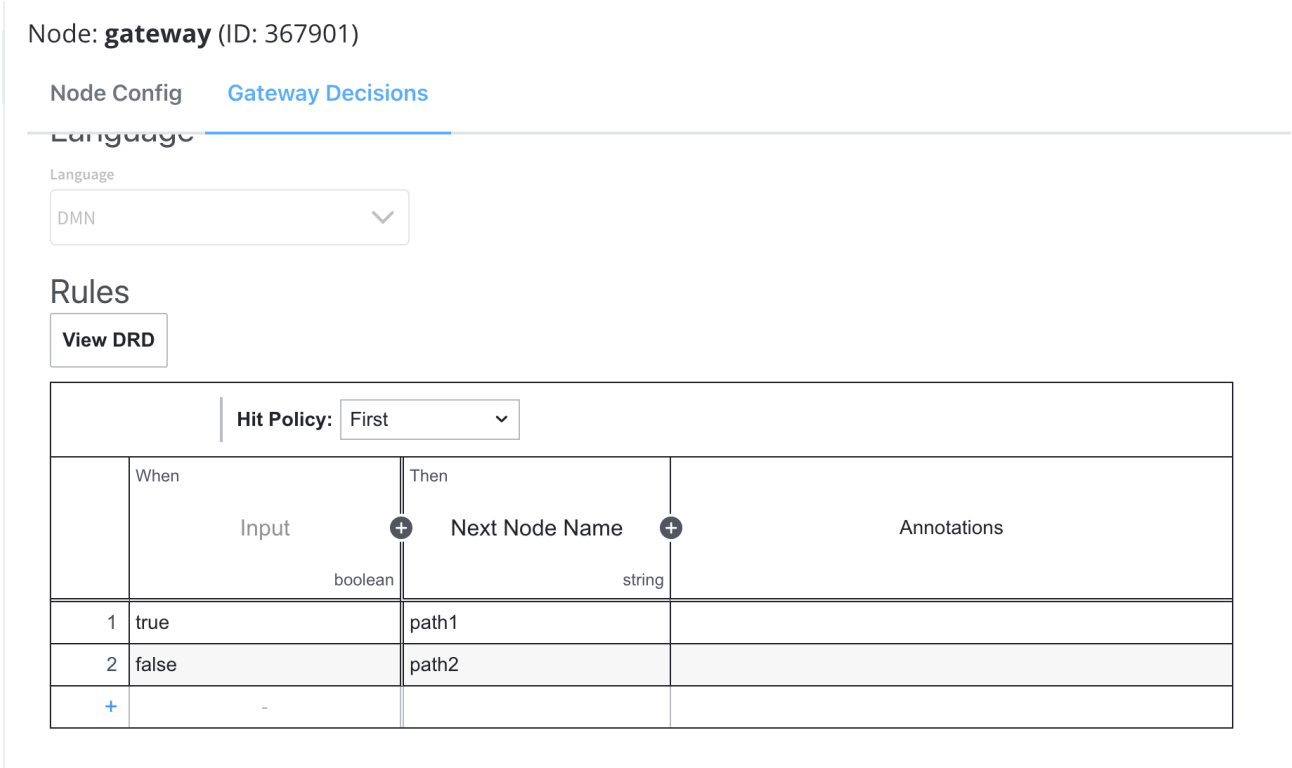


Configuring a DMN Exclusive Gateway node

You can use DMN to define gateway decisions, using exclusive gateways.



Gateway Decision - DMN example (applicable only for exclusive gateway - XOR)



Was this page helpful?

BUILDING BLOCKS / Node / Parallel gateway

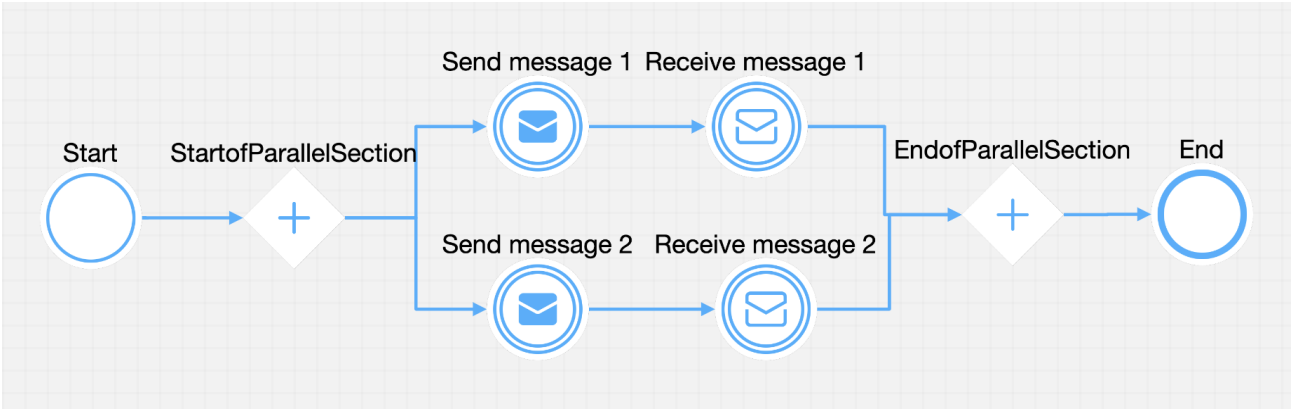
If multiple operations can be done in parallel a Parallel Gateway can be used. This kind of node will open a parallel section of the

The fallback content to display on prerendering
, very useful for integrations that can be done in parallel, without waiting for each other. Each parallel section should be also closed by another parallel Gateway node.

Configuring a Parallel gateway node

This kind of

The fallback content to display on prerendering
has no special configuration and can start 2 or more parallel paths. It is important to keep in mind that the close Parallel node, required to close the parallel section will wait for all branches to finish before moving to next node.



Was this page helpful?

BUILDING BLOCKS / Node / Milestone node

A **milestone node** is used to define how user tasks (which are placed between two milestones - **start milestone** and **end milestone**) will be displayed.

Multiple options are available for displaying the content:

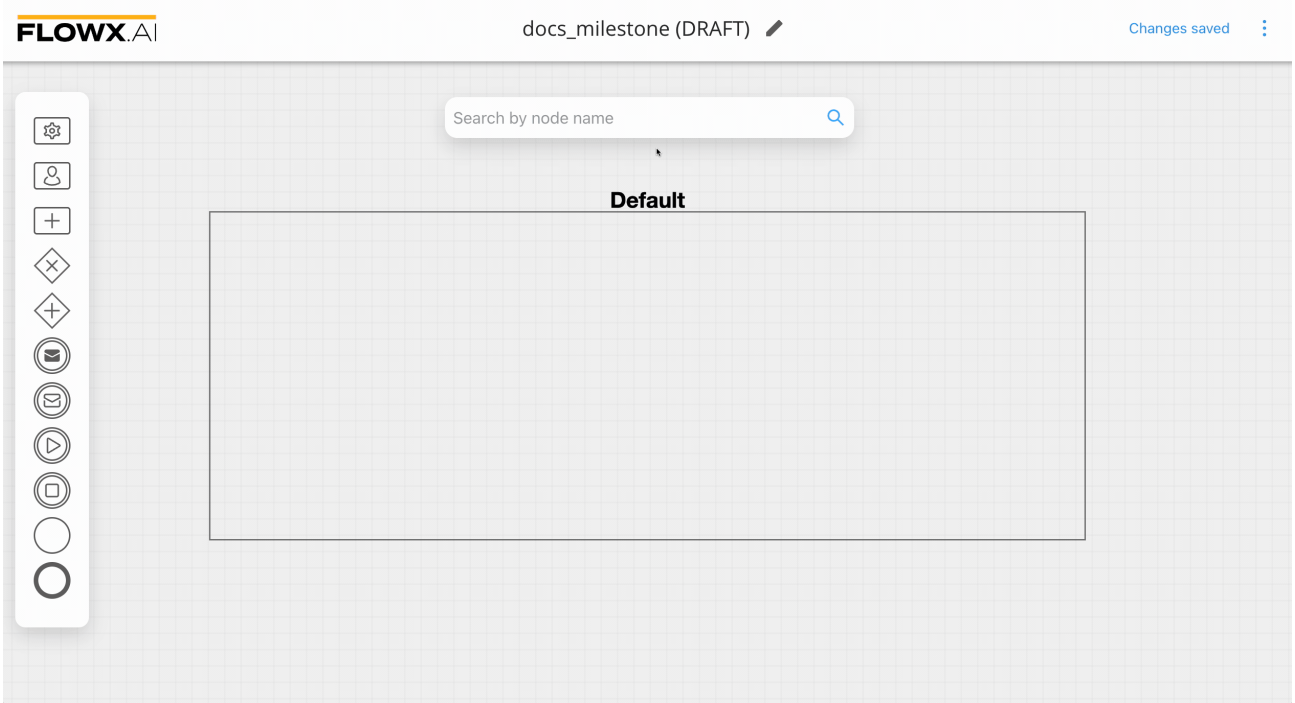
- Modal
- Page
- Stepper + Steps
- Container

Configuring a Milestone node

A combination of **start** and **end** nodes can be used to achieve all kinds of a grouping of multiple user task nodes.

To configure a user task to be displayed in a Modal:

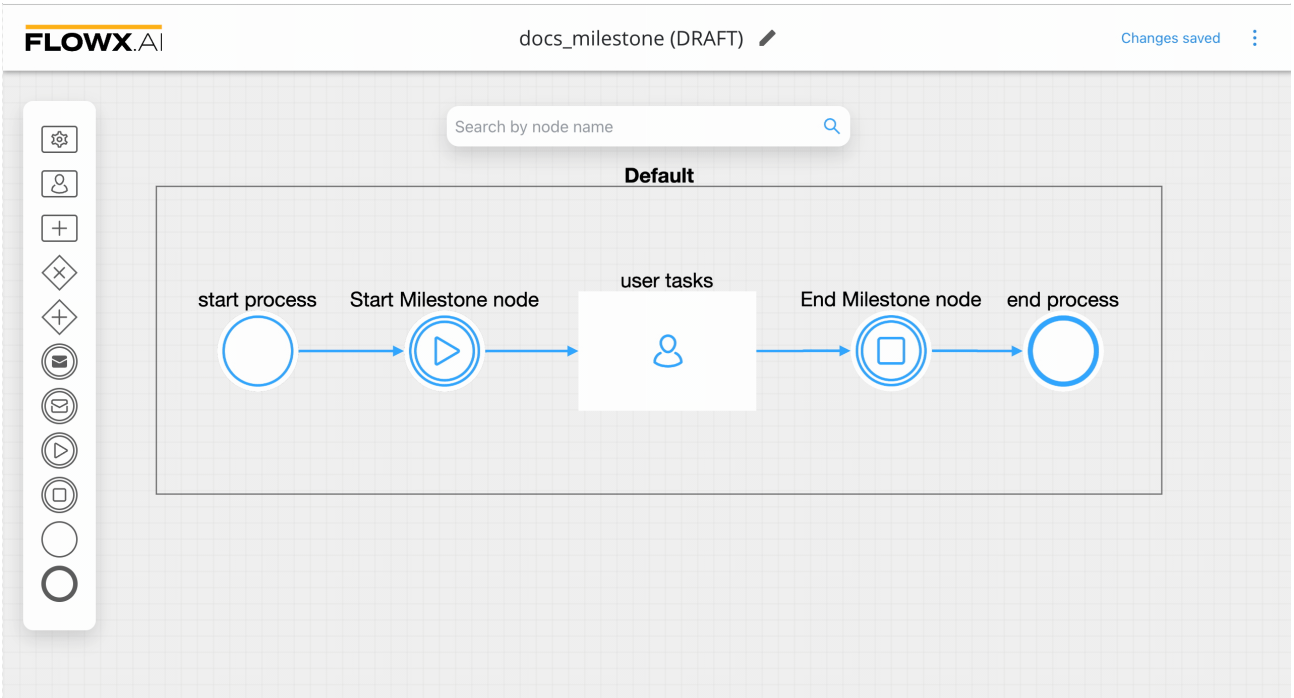
1. Open **Process Designer** and start configuring a process.
2. Add a **user task** that you want to display.
3. Add a **start milestone** before the user task.
4. Add an **end milestone** after the user task.



5. Select the **start milestone node** and open UI Designer - here you can choose from multiple templates of how to display the content.

6. For example, drag and drop the **modal** template to the canvas.

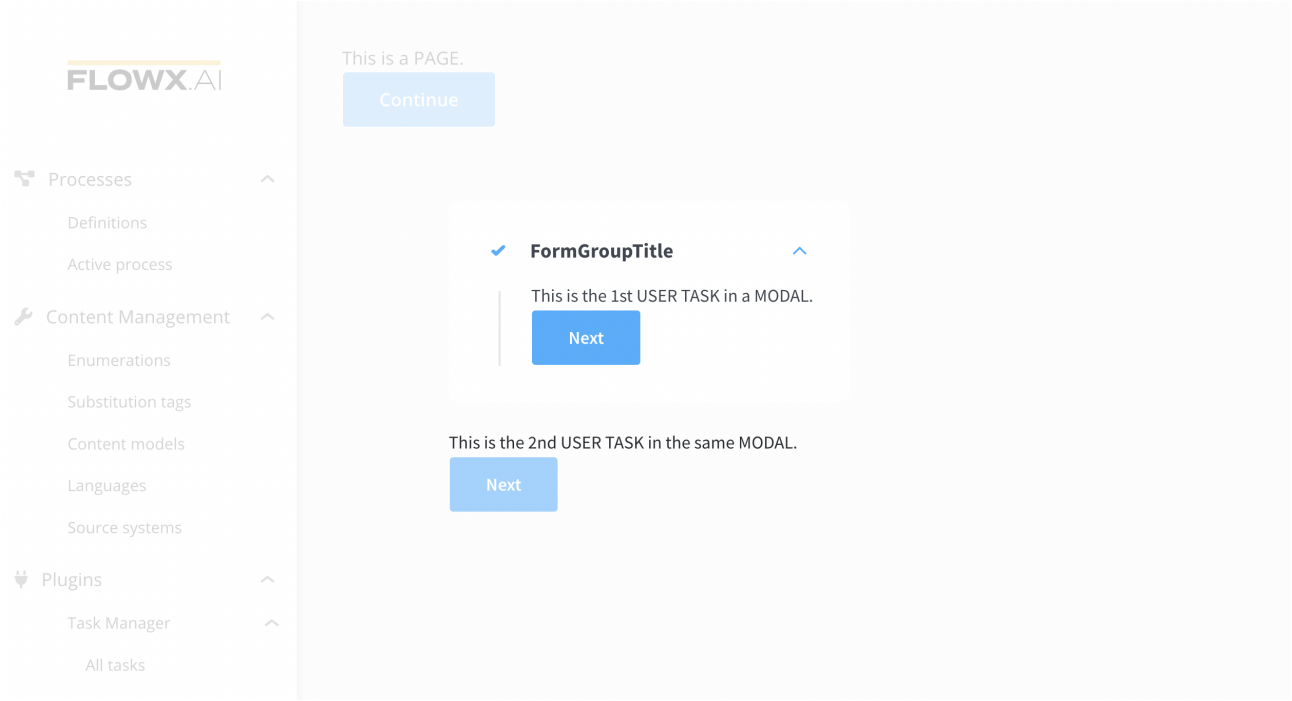
7. No additional information is required for displaying a **user task** in a modal view but you can do multiple customizations via the different configurations using the UI Designer.



Available Components

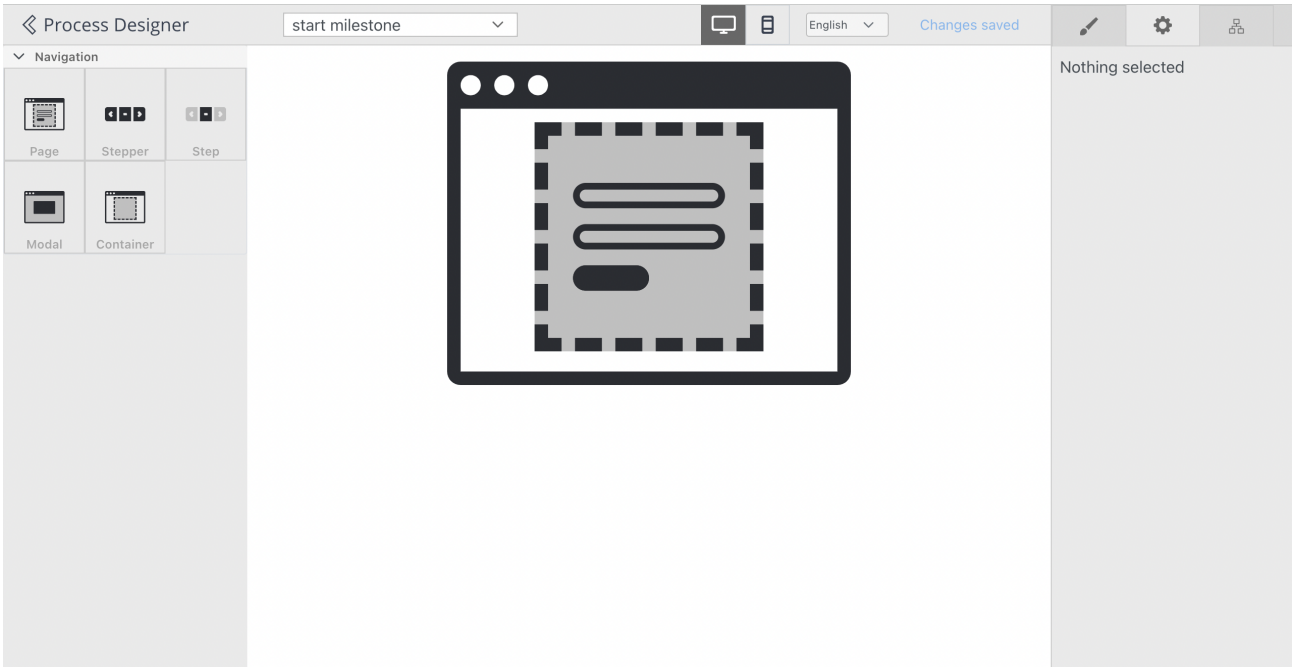
Modal

You can configure a start milestone node and an end milestone node before and after a user task. After adding the milestones, you can add a modal template to the start milestone node to display a modal screen (like in the example above).



Page

A basic full page content can be displayed using this kind of template on a milestone start.



Stepper + Steps

To create a stepper architecture:

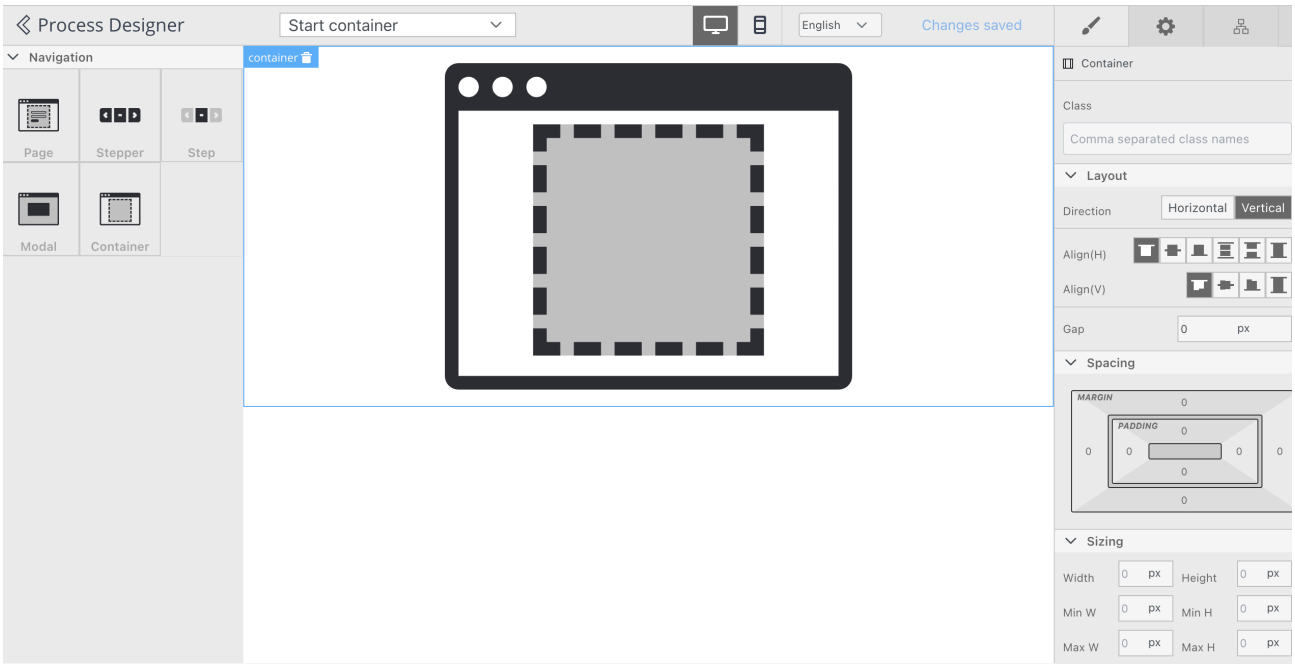
1. First define a **milestone start node**.
2. Then add a **Stepper template** on the first node (and a **milestone end** after the **first node**).
3. In between the **stepper milestones** add for each **step** a **milestone start** and a **milestone end** node with a **Step** configuration.

For more information about how to create a process with a Stepper + Steps structure, and how to configure the UI, check the following section:

» Create a User Interface

Container

Containers allows us to display multiple user task on the same Page/Modal/Step with a different layout, other than the basic one. You can use **Layout** tab to play with multiple alignments.



Was this page helpful?

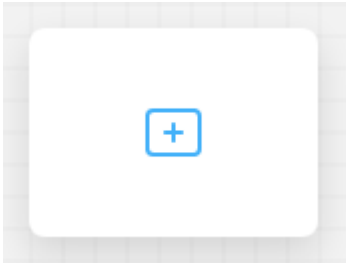
BUILDING BLOCKS / Node / Subprocess run node

> **INFO**
>
> There might be cases when extra functionality is needed on certain
>
> The fallback content to display on prerendering
>
> .

A node that provides advanced options for starting

The fallback content to display on prerendering

.



» Subprocess

It contains a default action for starting a subprocess.

A subprocess can be started in two modes:

- **async mode** - the parent
  The fallback content to display on prerendering
  will continue without waiting for the sub-process to finish

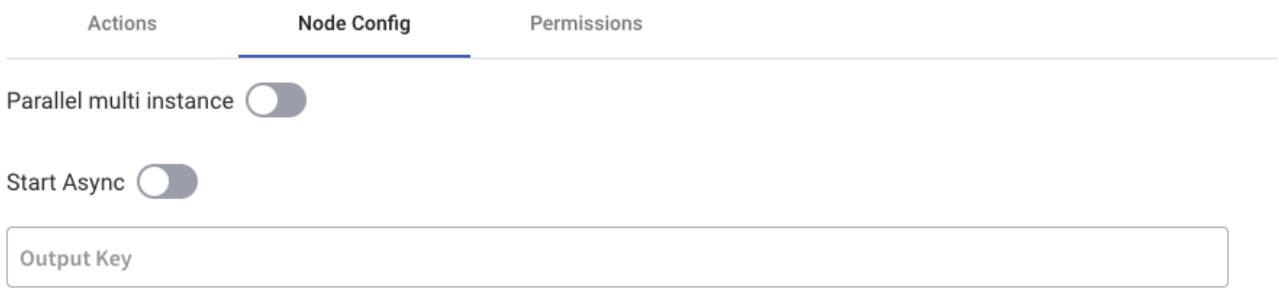
> **INFO**

> Select if this task should be invoked asynchronously. Make tasks asynchronous if they cannot be executed instantaneously, for example, a task performed by an outside service.

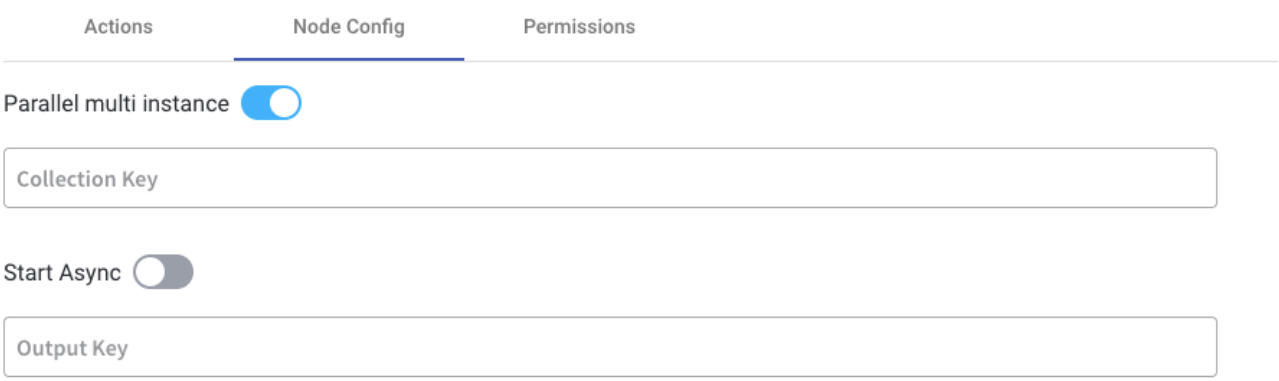
- **sync mode** - the parent process must wait for the subprocess to finish before advancing

The start mode can be chosen when configuring the subprocess run node.

In case the parent process needs to wait for the subprocess to finish and retrieve some results from it, the parent process key that will hold the results must be defined using the *output key* node config value.



This node type can also be used for starting a set of subprocesses that will be started and run at the same time. This will prove useful in case we have an array of values in the parent process parameters and we want to start a subprocess for each of the elements in that array.



In order to do this, we need to select the parallel multi instance option. The *collection key* name from the parent process also needs to be specified.

> **INFO**
>
> When designing such a subprocess that will be started in a loop, you need to keep in mind that the input value for the subprocess (that is, one of the values from the array in the parent process) will be stored in the subprocess parameter values under they key named *item*. This will have to be used inside the subprocess. If this subprocess produces any results, they should be stored under a key named *result* in order to be sent back to the parent process.

Was this page helpful?

BUILDING BLOCKS / Node / Message events / Message Throw Intermediate

Event

QUICK INTRO

What is it? It's like throwing a message to tell someone about something. After throwing the message, the process keeps going, and other parts of the process can listen to that message.

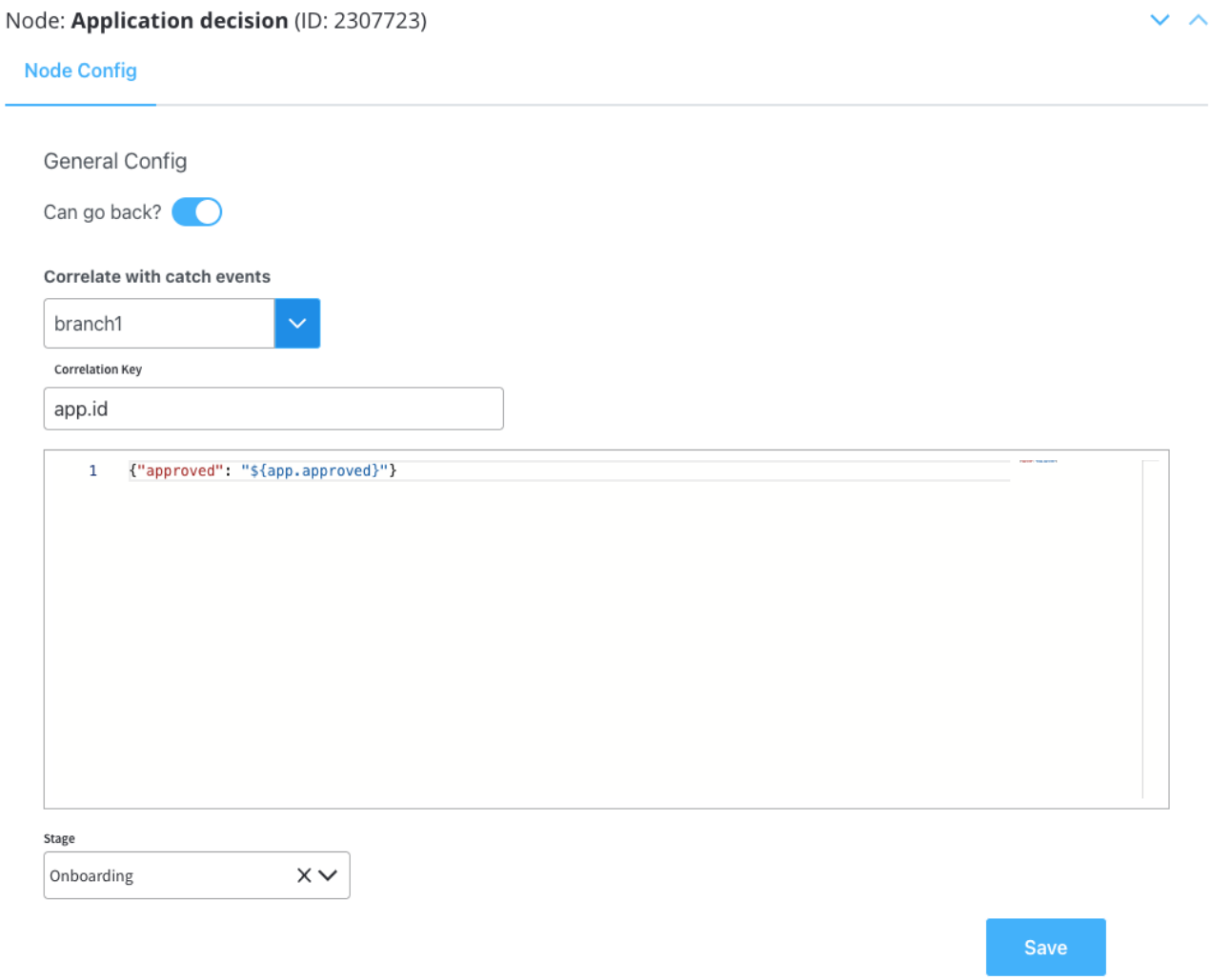
Why it is important? The Message Throw Intermediate Event is important because it allows different parts of a process to communicate and share information with each other.

Configuring a Message Throw Intermediate Event

A Message Throw Intermediate Event is an event in a process where a message is sent to trigger a communication or action with another part of the process (can be correlated with a catch event). It represents the act of throwing a message to initiate a specific task or notification. The event creates a connection between the sending and receiving components, allowing information or instructions to be transmitted. Once the message is thrown, the process continues its flow while expecting a response or further actions from the receiving component.

General config

- **Can go back?** - setting this to true will allow users to return to this step after completing it, when encountering a step with `canGoBack` false, all steps found behind it will become unavailable
- **Correlate with catch events** - the dropdown contains all catch messages from the process definitions accessible to the user
- **Correlation key** - is a process key that uniquely identifies the instance to which the message is sent
- **The data field** - allows the user to define a JSON structure with the data to be sent along with the message
- **Stage** - assign a stage to the node



Was this page helpful?

BUILDING BLOCKS / Node / Message events / Message Catch Boundary Events

QUICK INTRO

What is it? A Message Catch Boundary Event is a special

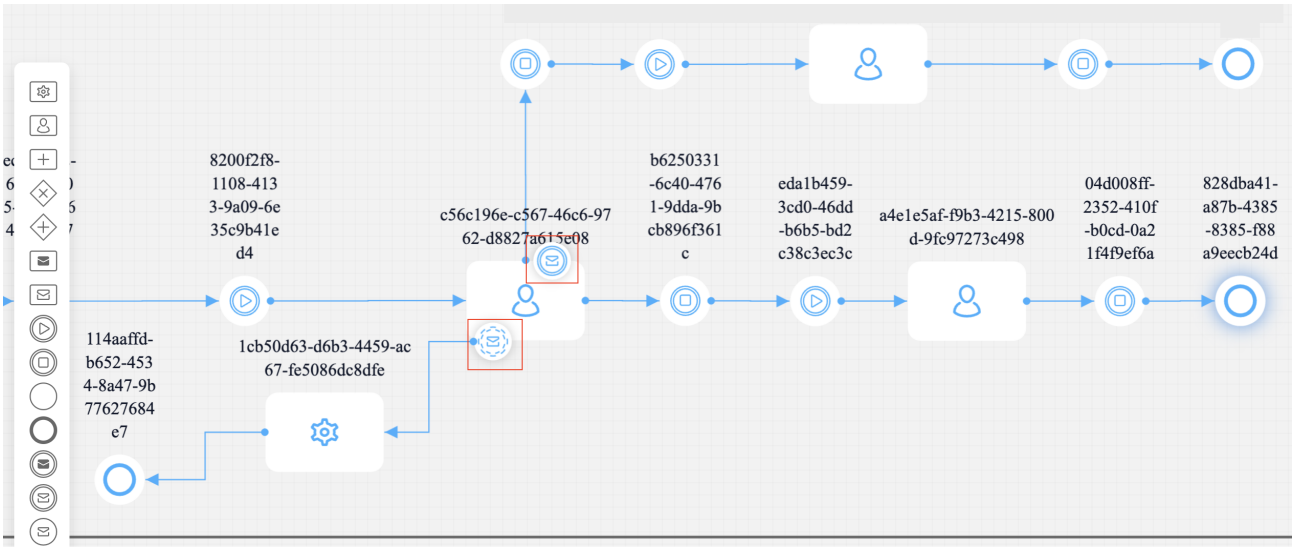
The fallback content to display on prerendering
that can be attached to a user task in a
The fallback content to display on prerendering
.

Why it is important? It allows the process to listen for and capture specific messages during the execution of the associated user task.

When used as a boundary event on a **user task**, message catch boundary event nodes behave similar to an **exclusive gateway**, but they are activated upon receiving an event. This means you can proceed in the process without receiving an event and continue through the sequence initiated from the user task.

If an event is received, it advances through the sequence from the intermediate

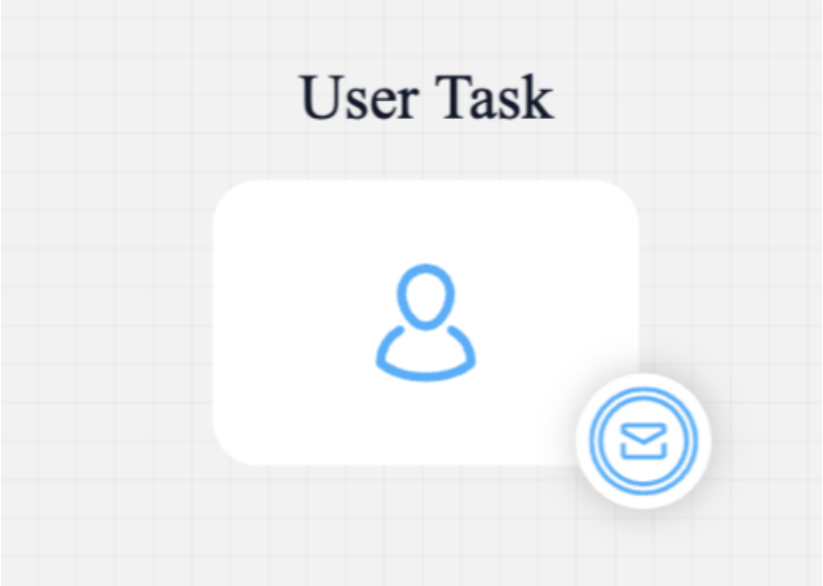
The fallback content to display on prerendering
. You can have multiple intermediate boundary events on the same user task, but only one can be activated at a time.



There are two types of Message Catch Boundary Events:

- **Interrupting**
- **Non-Interrupting**

Message Catch Interrupting Event



When an Interrupting Message Catch Boundary Event is triggered by receiving a message, it interrupts the associated task that is being performed. The task is immediately finished, and the

The fallback content to display on prerendering
continues to advance based on the received message.

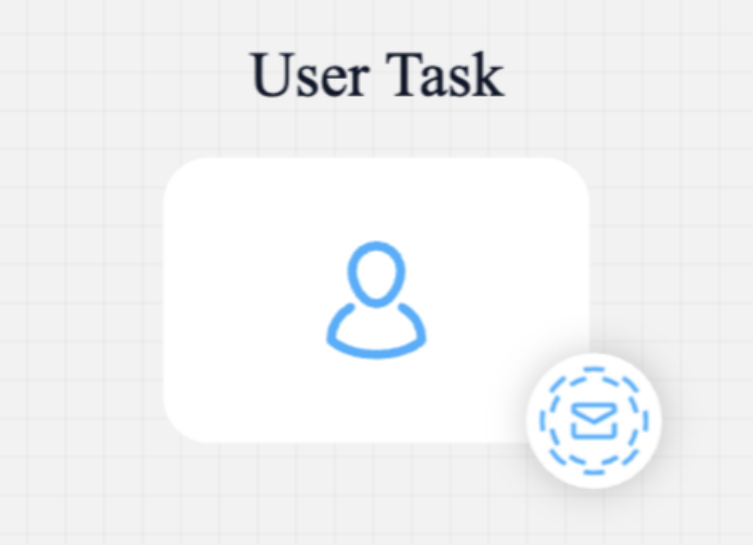
It can also be used as a standalone node, see more information on the following section:

» Message Catch Intermediate Event

Message Catch Non-Interrupting Event

It is used only as a boundary event and is placed only on a user task. If your process is in that user task and receives

The fallback content to display on prerendering
, the event is activated, and a new token is created that advances independently. Sections with non-interrupting events should not contain user tasks. You can have multiple non-interrupting events on the same user task, and all of them can be activated simultaneously.



A Non-Interrupting Message Catch Boundary Event also listens for messages while the associated task is being performed. However, in this case, the task is not immediately finished when messages are received. The event captures the messages, allowing the task to continue its execution. Multiple non-interrupting events can be received while the task is still active, and the task will continue until its completion.

Configuring a Message Catch Interrupting/Non-Interrupting Event

General config

- **Correlate with throwing events** - the dropdown contains all throw events from the process definitions accessible to the user

> **INFO**
>
> It is used to establish the correlation between the catch event and the corresponding throw event. By selecting the appropriate throw event, the catch event will be triggered when a message is thrown from that event.

- **Correlation key** - process key used to establish a correlation between the received message and a specific process instance

> **INFO**
>
> Correlation key serves as a means to correlate the incoming message with the specific process instance it belongs to. When a message is received with a matching correlation key, the catch event will be triggered.

- **Receive data (process key)** - the catch event can receive data associated with the message and store it in a process variable with the specified process key

> **INFO**

This data can then be used within the process instance for further processing or decision-making.

Was this page helpful?

BUILDING BLOCKS / Node / Message events / Message Catch Intermediate Event

QUICK INTRO

What is it? A Message Catch Intermediate Event is a type of event in a process that waits for a specific message before continuing with the process flow.

Why it is important? It enables the process to synchronize and control the flow based on the arrival of specific messages, ensuring proper coordination between process instances.

Similar to the Message Catch Boundary Event, the Message Catch Intermediate Event is important because it facilitates the communication and coordination between process instances through messages. By incorporating this event, the process can effectively synchronize and control the flow based on the arrival of specific messages.

INFO

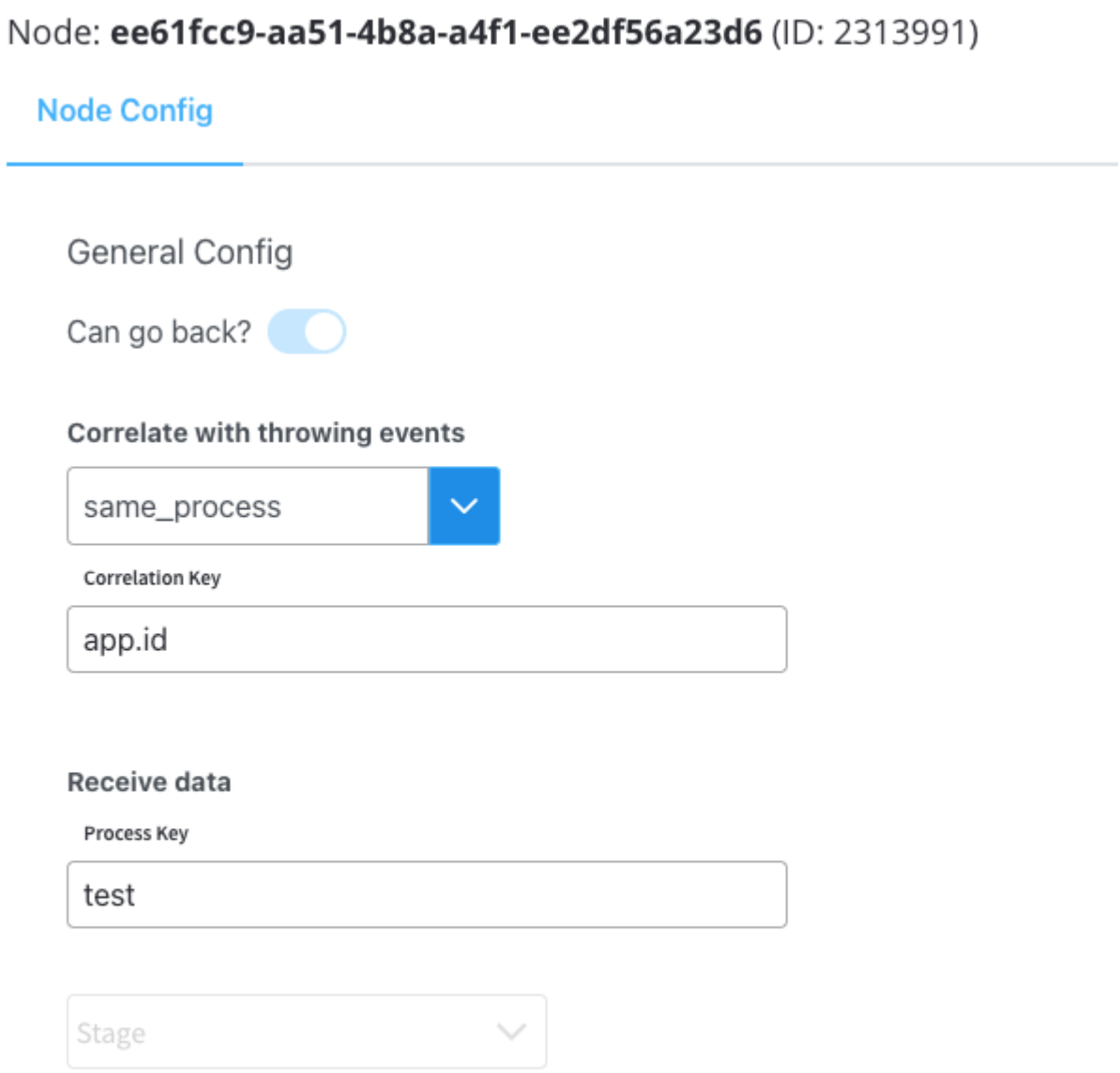
> Message Catch Intermediate Event can be used as a standalone node, this means that it will block a process until it receives an event.

Configuring a Message Catch Intermediate Event

Imagine a process where multiple tasks are executed in sequence, but the execution of a particular task depends on the arrival of a certain message. By incorporating a Message Catch Intermediate Event after the preceding task, the process will pause until the expected message is received. This ensures that the subsequent task is not executed prematurely and allows for the synchronization of events within the process.

General config

- **Can go back?** - setting this to true will allow users to return to this step after completing it, when encountering a step with `canGoBack` false, all steps found behind it will become unavailable
- **Correlate with throwing events** - the dropdown contains all catch messages from the process definitions accessible to the user
- **Correlation key** - process key used to establish a correlation between the received message and a specific process instance
- **Receive data** - the process key that will be used to store the data received along with the message
- **Stage** - assign a stage to the node



Was this page helpful?

BUILDING BLOCKS / Node / Message events / Message Catch Start Event

QUICK INTRO

What is it? It represents the starting point for a process instance based on the receipt of a specific message. When this event is triggered by receiving the designated message, it initiates the execution of the associated process.

Why it is important? The Message Catch Start Event is important because it allows a process to be triggered and initiated based on the reception of a specific message.

Configuring a Message Catch Start Event

A Message Catch Start Event is a special event in a process that initiates the start of a process instance upon receiving a specific message. It acts as the trigger for the process, waiting for the designated message to arrive. Once the message is received, the process instance is created and begins its execution, following the defined process flow from that point onwards. The Message Catch Start Event serves as the entry point for the process, enabling it to start based on the occurrence of the expected message.

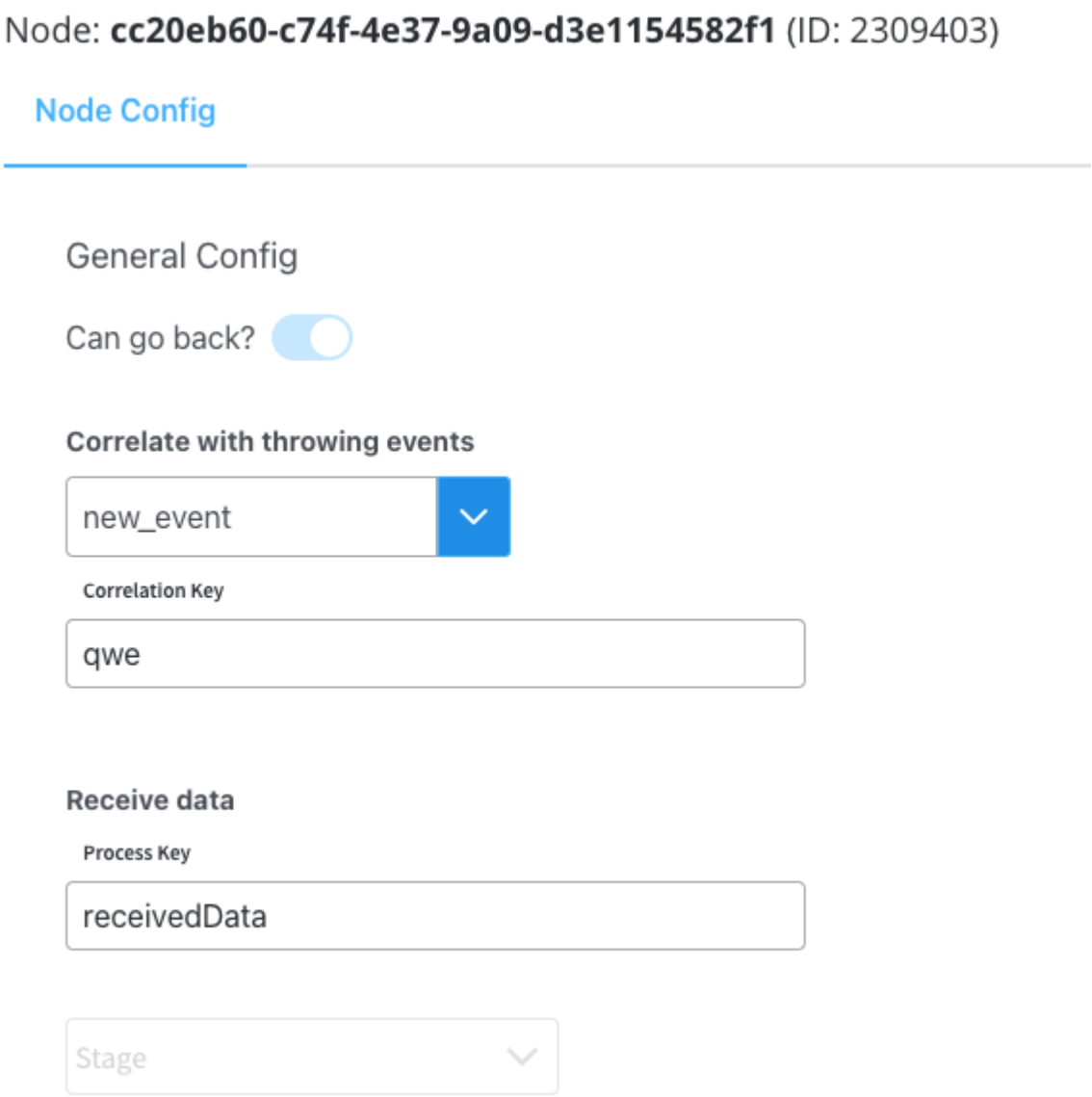
CAUTION

It is mandatory that in order to use this type of node together with task management plugin, to have a service account defined in your identity

> solution. For more information, check our documentation in how to create service accounts using Keycloak, **here**

General config

- **Can go back?** - setting this to true will allow users to return to this step after completing it, when encountering a step with `canGoBack` false, all steps found behind it will become unavailable
- **Correlate with catch events** - the dropdown contains all catch messages from the process definitions accessible to the user
- **Correlation key** - is a process key that uniquely identifies the instance to which the message is sent
- **The data field** - allows the user to define a JSON structure with the data to be sent along with the message
- **Stage** - assign a stage to the node



Was this page helpful?