
FLOW^X.AI

BUILDING BLOCKS

Contents

- BUILDING BLOCKS / Process Designer / Process definition
 - History
 - Versions
 - Audit log
 - Data model
 - Attributes type
 - Data model reference
 - Sensitive data
 - Reporting
 - Generating data model
 - Swimlanes
 - Settings
 - General
 - Permissions
 - Task management
- BUILDING BLOCKS / Process Designer / Active process / Process instance
 - Overview
 - Checking the Process Status
 - Understanding the Process Status Data
 - Process menu
 - Color coding
 - Starting a new process instance
 - Troubleshooting possible errors
- BUILDING BLOCKS / Process Designer / Active process / Failed process start
 - Exceptions
 - Exceptions data

- Exceptions type
- BUILDING BLOCKS / Process Designer / Subprocess
 - Configuring & starting subprocesses
 - Executing subprocesses
- BUILDING BLOCKS / Node / Start/End nodes
 - Start node
 - Configuring a start node
 - End node
 - Configuring an end node
- BUILDING BLOCKS / Node / Message send/Message received task nodes
 - Message send task
 - Configuring a message send task node
 - Example of a message send event
 - Message receive task
 - Configuring a message receive task node
- BUILDING BLOCKS / Node / Task node
 - Configuring task nodes
 - Configuring task nodes actions
 - Business Rule action
 - Send data to user interface
 - Upload File action
 - Start Subprocess action
 - Append Params to Parent Process
- BUILDING BLOCKS / Node / User task node
 - Configuring a user task node
 - Configuring the UI
 - Accessing the UI Designer
 - Predefined components
 - Custom components

- Displaying a UI element
- Values
- BUILDING BLOCKS / Node / Exclusive gateway
 - Configuring an Exclusive gateway node
- BUILDING BLOCKS / Node / Parallel gateway
 - Configuring a Parallel gateway node
- BUILDING BLOCKS / Node / Milestone node
 - Configuring a Milestone node
 - Available Components
 - Modal
 - Page
 - Stepper + Steps
 - Container
- BUILDING BLOCKS / Node / Subprocess run node
- BUILDING BLOCKS / Node / Message events / Message Throw Intermediate Event
 - Configuring a Message Throw Intermediate Event
- BUILDING BLOCKS / Node / Message events / Message Catch Boundary Events
 - Message Catch Interrupting Event
 - Message Catch Non-Interrupting Event
 - Configuring a Message Catch Interrupting/Non-Interrupting Event
- BUILDING BLOCKS / Node / Message events / Message Catch Intermediate Event
 - Configuring a Message Catch Intermediate Event
- BUILDING BLOCKS / Node / Message events / Message Catch Start Event
 - Configuring a Message Catch Start Event
- 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 / UI Designer / UI component types / Root components / Container
- BUILDING BLOCKS / UI Designer / UI component types / Root components / Card
- BUILDING BLOCKS / UI Designer / UI component types / Root components / Custom
- BUILDING BLOCKS / UI Designer / UI component types / Collection / Collection Prototype
 - Description
 - Configurable properties:
 - Example
 - Adding elements with UI Actions
 - Step 1 - Defining the Node Action
 - Step 2 - Adding the Button & UI Action
 - Result
- BUILDING BLOCKS / UI Designer / UI component types / Buttons
 - Basic button
 - Configuring a basic button
 - Button styling

- File upload
 - Configuring a file upload button
 - Button styling
- BUILDING BLOCKS / UI Designer / UI component types / File Preview
 - Configuring a File Preview element
 - File Preview properties (web)
 - File Preview properties (mobile)
 - File preview styling
 - File Preview example
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Input
 - Configuring the input element
 - Input settings
 - Input styling
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Text area
 - Configuring the text area element
 - Text area settings
 - Text area styling
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Select
 - Configuring the Select element
 - Select Settings
 - Select styling
 - Example - Dynamic dropdowns
 - Creating the process
 - Configuring the nodes
 - Configuring the UI

- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Checkbox
 - Configuring the checkbox element
 - Checkbox settings
 - Checkbox styling
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Radio
 - Configuring the radio field element
 - Radio settings
 - Radio styling
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Switch
 - Configuring the switch element
 - Switch settings
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Datepicker
 - Configuring the datepicker element
 - Datepicker settings
 - Datepicker styling
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Slider
 - Configuring the slider element
 - Slider settings
 - Multiple sliders
 - Slider styling
- BUILDING BLOCKS / UI Designer / UI component types / Form elements / Segmented button
 - Configuring the segmented button
 - Segmented button settings

- Segmented button styling
- BUILDING BLOCKS / UI Designer / UI component types / Image
 - Configuring an image
 - Image settings
 - Media library
 - Process Data
 - External
 - UI actions
 - Image styling
- BUILDING BLOCKS / UI Designer / UI actions
 - Process UI actions
 - Manual action configuration example - Save Data
 - UI action configuration example
 - UI actions elements
 - Events
 - Action types
 - External UI actions
- BUILDING BLOCKS / UI Designer / Validators
 - Predefined validators
 - required validator
 - minlength validator
 - maxlength validator
 - min validator
 - max validator
 - email validator
 - pattern validator
 - datepicker - isSameOrBeforeToday
 - datepicker - isSameOrAfterToday
 - Custom validators

- sidebar_position: 3
- BUILDING BLOCKS / UI Designer / Dynamic & computed values
 - Dynamic values
 - Example using Substitution tags
 - Example using process parameters
 - Computed values
 - Slider example
 - Usage
- BUILDING BLOCKS / UI Designer / Layout configuration
- BUILDING BLOCKS / UI Designer / Rendering and UI Designer changelog
 - Notes for post-migration
- BUILDING BLOCKS / Token
- BUILDING BLOCKS / Supported scripts
 - Supported scripts
 - Python
 - DMN
 - MVEL
 - Groovy
 - Nashorn Engine (JavaScript)

BUILDING BLOCKS / Process Designer / Process definition

The core of the platform is the process definition, which is the blueprint of the business process made up of **nodes** that are linked by sequences.

Process Definitions

⋮

Drafts / In progress

Name	Version	Edited at	Edited by	⋮
Amazing Process	1	05 Oct 2022, 4:21 PM	John Doe	▶ ✍ ⋮
Awesome Process	1	05 Oct 2022, 4:07 PM	John Doe	▶ ✍ ⋮
Exquisite Process	2	05 Oct 2022, 2:49 PM	Jane Doe	▶ ✍ ⋮

Published

Name	Version	Published at	Published by	⋮
Incredible Process	1	05 Oct 2022, 2:49 PM	John Doe	▶ ✍ ⋮
Breathtaking Process	1	05 Oct 2022, 8:11 AM	Bess Twishes	▶ ✍ ⋮
Stunning Process	1	04 Oct 2022, 12:30 PM	Silviu Grigore	▶ ✍ ⋮

Once a process is defined and set as published on the platform, it can be executed, monitored, and optimized. When a business process is started, a new instance of the definition is created.

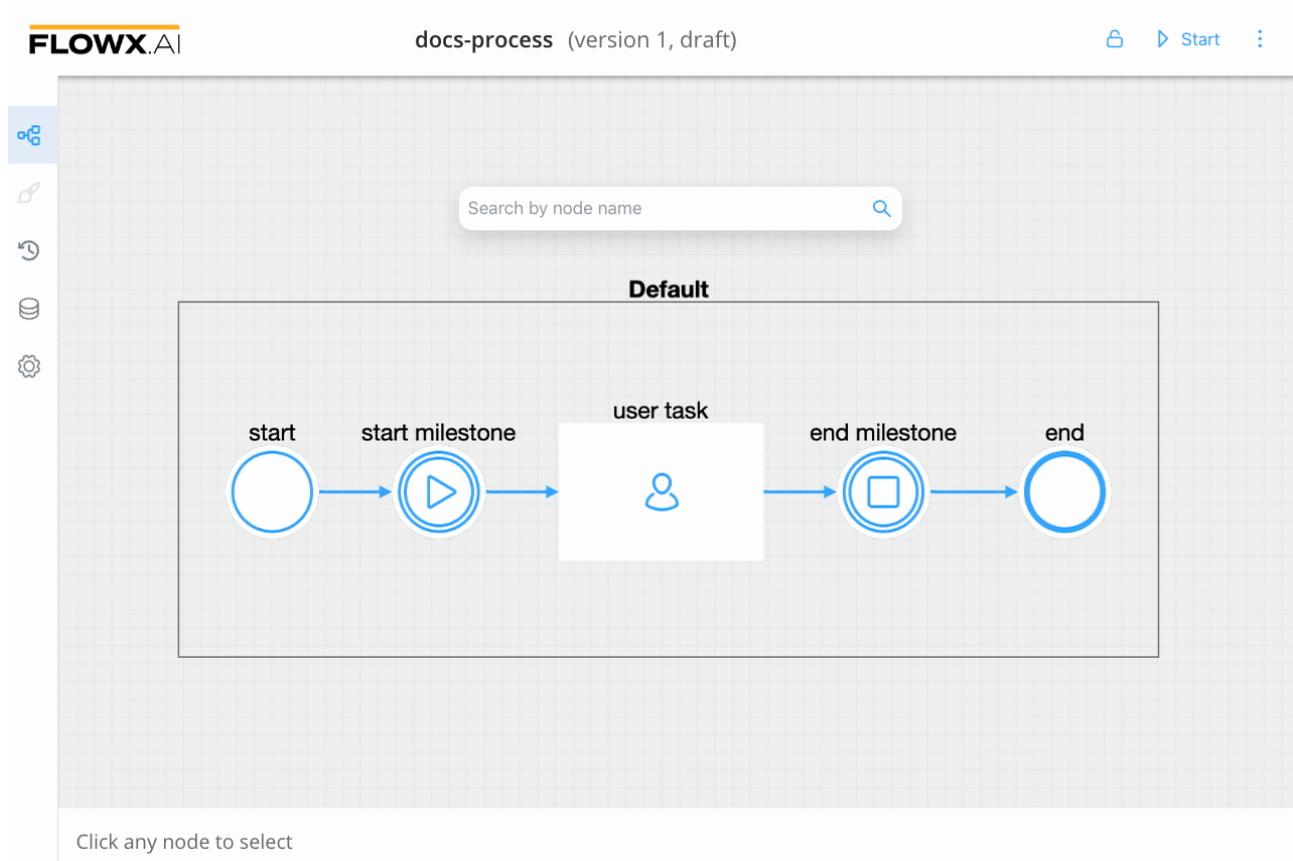
» [Process instance](#)

» [Failed process start](#)

History

In the **History** tab, you will find a record of all the modifications and events that have occurred in the process.

- **Versions** - provides information on who edited the process, when it was modified, and the version number and status
- **Audit log** - provides a detailed record of events and changes



Versions

In the **Versions** tab you will find the following details:

- Last edited on - the last time when the process was modified
- Last edited by - the last person who modified a process

- Version - version number
- Status - can be either **Published** or **Draft**

! HINT

! Published processes cannot be modified (they must be deprecated to be set as **Draft** before editing them).

- View process - clicking on the eye icon will redirect you to the process definition

Audit log

In the **Audit log** tab you will find the following items:

- Timestamp
- User
- Subject
- Event
- Subject Identifier
- Version
- Status

! INFO

Some items in the Audit log are filterable, making it easy to track changes in the process.

» Audit

Data model

In the Data Model, you can add new key-pair values, which enables you to use shortcuts when adding new keys using the UI Designer, without having to switch back and forth between menus.

X generate_data_model | Version 1 | draft

+ New item ⏷ ⏸ Start ⋮

Data Model				
	Name	Type	Used in reporting	Description
⌚	application	OBJECT	-	no + ⎯ ⚡
⌚	client	OBJECT	-	no + ⎯ ⚡
⌚	otherDeclarations	OBJECT	-	no + ⎯ ⚡
⌚	identificationData	OBJECT	-	no + ⎯ ⚡
⌚	legalAddress	OBJECT	-	yes + ⎯ ⚡
⌚	residenceAddress	OBJECT	-	no + ⎯ ⚡
⌚	mailingAddress	OBJECT	-	no + ⎯ ⚡
	lastName	STRING	yes	no ⎯ ⚡
	firstName	STRING	yes	no ⎯ ⚡

Attributes type

Add attribute

Type: STRING

Attribute name: ^

! Name is required

Description:

Example value:

Use in reporting

Sensitive Data

i Sensitive data will be hidden for unauthorised personnel. More info [here](#)

Close Save

The Data Model supports the following attribute types:

- STRING
- NUMBER
- BOOLEAN
- OBJECT

- ARRAY
 - ARRAY OF STRINGS
 - ARRAY OF NUMBERS
 - ARRAY OF BOOLEANS
 - ARRAY OF OBJECTS
 - ARRAY OF ENUMS
- ENUM

 INFO

When you export or import a **process definition**, the data model will be included.

Data model reference

You can use data model reference feature to view attribute usage within the data model. You can now easily see where a specific attribute is being used by accessing the "View References" feature. This feature provides a list of process keys associated with each attribute and displays possible references, such as UI Elements.

For UI Elements, the references include the element label, node name, and UI Element key. Additionally, the context of the reference is provided, showing the node name and the UI element type along with its label. Users can conveniently navigate to the context by clicking the provided link to the node's UI page.

The screenshot shows the FLOWX.AI Data Model editor interface. At the top, there's a header with a 'data_model' title, a 'Version 1' badge, and a 'draft' status. To the right are buttons for '+ New item', 'Start', and a three-dot menu. On the left, there's a vertical toolbar with icons for creating, deleting, editing, and more. The main area is a table titled 'Data Model' with the following columns: Name, Type, Used in reporting, Sensitive data, and Description. The table contains the following data:

Name	Type	Used in reporting	Sensitive data	Description
firstName	STRING	no	no	
lastName	STRING	no	no	
dateOfBirth	STRING	no	no	
segmentedKey	STRING	no	no	
switch	BOOLEAN	no	no	
> test_object	OBJECT	no	no	

Sensitive data

To protect your data and your customer's data, you can hide data that could be visible in the process details or in the browser's console. You can now also secret data for a specific key.

X

Add Attribute

Attribute name**Type**

street

Placeholder

**Enumeration (Opt)**

Placeholder

**Example Value**

Placeholder

Description (Opt)

first steps of the process without the br

**Use in reporting****Sensitive data**

Sensitive data will be hidden for unauthorised personnel. More info [here](#)

[Cancel](#)[Save](#)

Reporting

The **Use in Reporting** tag is used for keys that will be used further in the reporting plugin.

» **Reporting**

Generating data model

A data model can be generated using data values from a **process instance**. This can be done by either merging the data model with an existing one or replacing it entirely.

To generate a data model, follow these steps:

1. Open **FLOWX.AI Designer**.
2. Go to the **Definitions** tab and select the desired **process definition**.
3. Select the **Data Model** tab and then click **Generate data model** button.
4. Add the **process instance** of the process from which you want to generate the data model.
5. Choose whether to **replace** the existing data model or **merge** it with the new one.
6. Click the **Load Data** button to display the data model body.
7. Finally, click **Save** button to save the generated data model.

The screenshot shows the FLOWX.AI interface with a sidebar on the left containing icons for Home, Data Model, Flows, Reports, and Settings. The main area is titled "generate_data_model | Version 1 | draft". It displays a table with columns: Name, Type, Used in reporting, Sensitive data, and Description. A message "No attributes found" is shown below the table. At the top right, there are buttons for "New item", "Start", and a more options menu.

By generating a data model, you can ensure that your data is structured and organized in a way that is appropriate for your business needs. It can also help you to identify any inconsistencies or errors in the data, allowing you to correct them before they cause problems down the line.

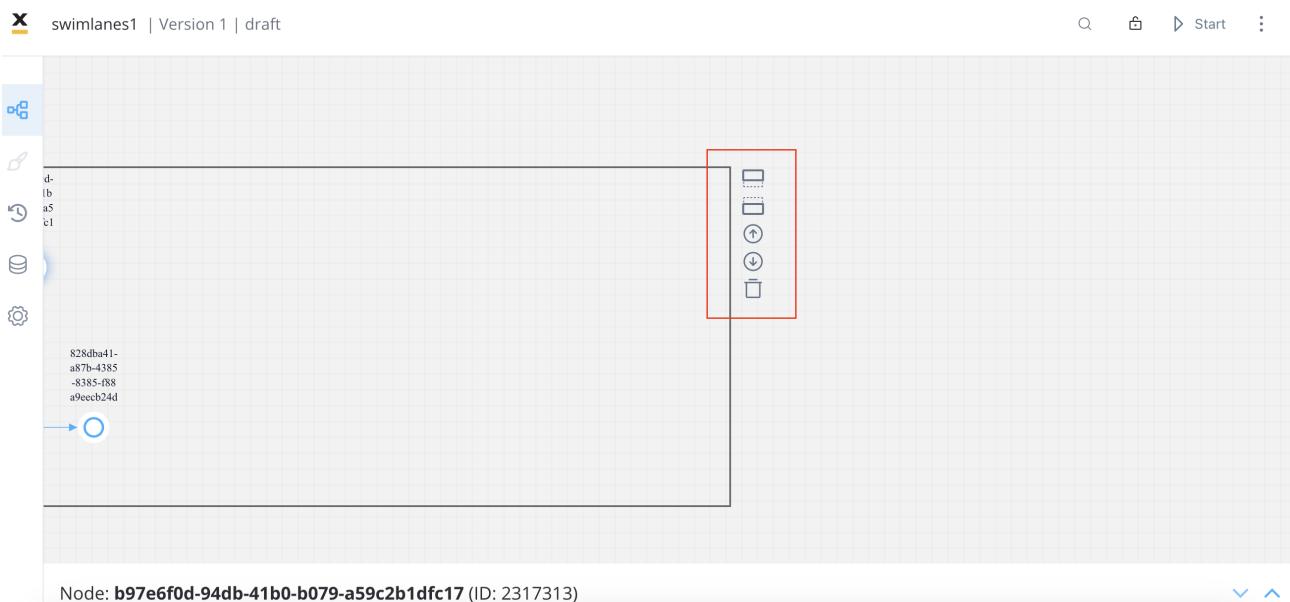
Swimlanes

Swimlanes offer a useful method of organizing process nodes based on process participants. By utilizing swimlanes, you can establish controlled access to specific process nodes for particular user roles.

Adding new swimlanes

To add new swimlanes, please follow these steps:

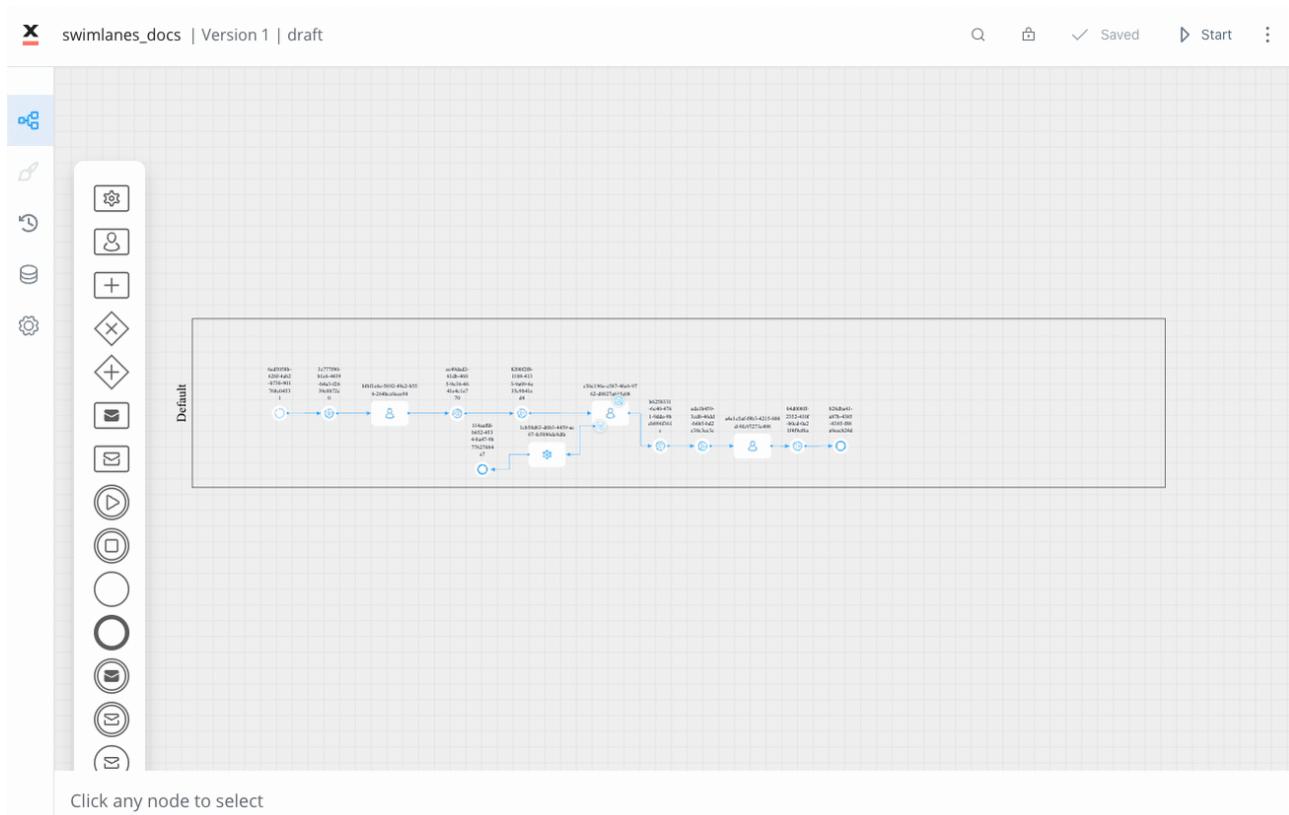
1. Access the **FLOWX.AI Designer**.
2. Open an existing process definition or create a new one.
3. Identify the default swimlane and select it to display the contextual menu.



INFO

With the contextual menu, you can easily perform various actions related to swimlanes, such as adding or removing swimlanes or reordering them.

4. Choose the desired location for the new swimlane, either below or above the default swimlane.
5. Locate and click the **add swimlane icon** to create the new swimlane.



For more details about user roles management, check the following section:

» [User roles management - Swimlanes](#)

For more details about setting up user role-based access on process definitions, check the following section:

» [Configuring access roles for processes](#)

Settings

General

In the General settings, you can edit the process definition name, include the process in reporting, set general data, and configure expiry time using Cron Expressions and ISO 8601 formatting.

- **Process definition name** - edit process definition name
- **Use process in reporting** - if switched on, the process will be included in reporting
- **Use process in task management** - if switched on, tasks will be created and displayed in the Task manager plugin, more information [here](#)
- **General data** - data that you can set and receive on a response
- **Expiry time** - a user can set up a `expiryTime` function on a process, for example, a delay of 30s will be set up like: `30 16 11 4 7 1`

For more information about **Cron Expressions** and **ISO 8601** formatting, check the following section;

» [Timer Expressions](#)

FLOWX.AI test-process (version 1, draft) [Start](#) [⋮](#)

The screenshot shows the 'General' tab selected in the top navigation bar. The 'Process definition name' is set to 'test-process'. A toggle switch labeled 'Use process in reporting' is turned off. A large text area for 'General data' is empty, containing only the number '1'. Below this is an 'Expiry time' input field with placeholder text 'ex: PT3M22S or 0 0 9-17 * * MON-FRI'. A note below the input field provides instructions for Cron Expression formatting. At the bottom right is a blue 'Save settings' button.

General* Sensitive data Swimlanes Permissions Task management

Process definition name

test-process

Use process in reporting

General data

1

Expiry time

ex: PT3M22S or 0 0 9-17 * * MON-FRI

For more details about Expiry time formatting and examples of valid Cron Expressions, click [here](#).

Save settings

Permissions

After defining roles in the identity provider solution, they will be available to be used in the process definition settings panel for configuring swimlane access.

When you create a new swimlane, it comes with two default permissions assigned based on a specific role: execute and self-assign. Other permissions can be added manually, depending on the needs of the user.

The screenshot shows the 'Permissions' tab of a process definition in the FlowX AI interface. On the left, there's a sidebar with 'Choose swimlane' and a list of 'docs 3.0' with roles 'SWIMLANE_USER' and 'SWIMLANE_SUPERVISOR'. The main panel shows the 'Default' swimlane configuration. It lists two roles: 'FLOWX_ROLE' with permissions 'Execute, Self Assign' and 'ROLE_ADMIN' with permissions 'View, Execute, Self Assign, Unassign, Assign'. There are 'Add Role' and 'Save permissions' buttons at the bottom.

» [Configuring access rights for processes](#)

Task management

The Task Management plugin offers a business-oriented view of the process you defined in the Designer and allows for interactions at the assignment level. It also

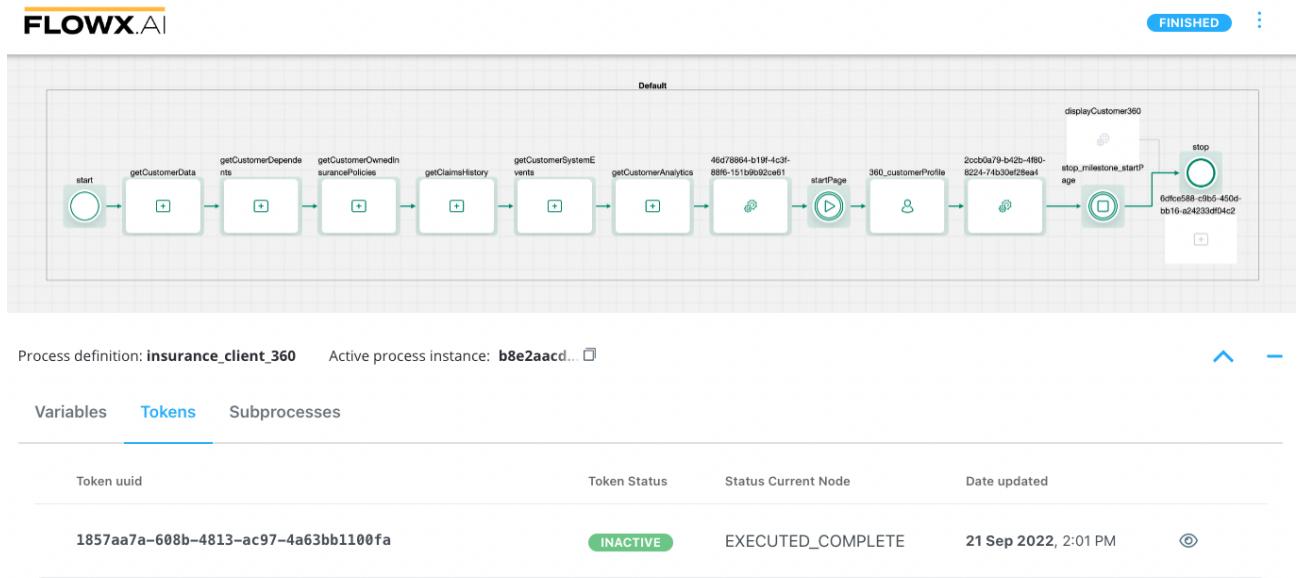
includes a generic parameter pointing to the application URL where the Flowx process is loaded and uses process keys to search data stored in the process.

The screenshot shows a sidebar with icons for General, Swimlanes, Permissions, and Task management. The Task management tab is selected. A main panel displays an 'Application url' field with a tooltip: 'Generic parameter pointing to the application url where the flowx process is loaded.' Below it is a 'Search indexing' section with a tooltip: 'Process keys will be used to search data stored in the process.' An 'Add Key' button is also visible.

Was this page helpful?

BUILDING BLOCKS / Process Designer / Active process / Process instance

A process instance is a specific execution of a business process that is defined on the FLOWX.AI platform. Once a process definition is added to the platform, it can be executed, monitored, and optimized by creating an instance of the definition.



Overview

Once the desired processes are defined in the platform, they are ready to be used. Each time a process needs to be used, for example each time a customer wants to request, for example, a new credit card, a new instance of the specified process definition is started in the platform. Think of the process definition as a blueprint for a house, and of the process instance as each house of that type being built.

The **FLOWX Engine** is responsible for executing the steps in the process definition and handling all the business logic. The token represents the current position in the process and moves from one node to the next based on the sequences and rules defined in the exclusive gateways. In the case of parallel gateways, child tokens are created and eventually merged back into the parent token.

Kafka events are used for communication between FLOWX.AI components such as the engine and integrations/plugins. Each event type is associated with a Kafka

topic to track and orchestrate the messages sent on Kafka. The engine updates the UI by sending messages through sockets.

» More about Kafka

Checking the Process Status

To check the status of a process or troubleshoot a failed process, follow these steps:

1. Open **FLOWX Designer**.
2. Go to **Processes** → **Active Process** → **Process instances**.
3. Click **Process status** button.

Active processes					
Process uuid	Definition name	Exceptions	Status	Current Node Name	Start date
cb19bfc0-cacb-4cbe-96bc-73c5771c0699	Amazing Process	-	CREATED	bddd53ae-125e-4f46-8288-6beae4424219	09 Aug 2022, 9:36 PM
abc17f2b-f3fb-48e1-a024-131692ade558	Super Process	-	STARTED	node_name	09 Aug 2022, 8:35 PM
3a4849fb-85ec-4383-9402-6f14d37d840e	Awesome Process	-	STARTED	user_task	09 Aug 2022, 8:32 PM
6989a30a-370a-4dc8-9aa8-e605bbe9772e	Incredible Process	-	STARTED	user_task	09 Aug 2022, 8:30 PM
4a69ff3c-6a48-46c6-9292-877f9bd7cf69	Nice Process	-	EXPIRED	page_1	09 Aug 2022, 7:47 PM

Understanding the Process Status Data

The process status data includes the following:

The screenshot shows a process instance details page for an 'Amazing Process'. At the top, there's a process diagram with four states: a red circle, a play button, a square, and a plus sign. Below the diagram, the process definition name is 'Amazing Process' and the active process instance ID is 'cb19bfc0...'. A modal window on the right displays the status 'CREATED'. Underneath the process details, there are tabs for 'Variables', 'Tokens', and 'Subprocesses'. The 'Variables' tab lists several variables with their values:

```
processInstanceId: 620353
tokenId: 620403
tokenUuid: "f3e76161-3ebc-4dd5-8baf-a921bc499126"
webSocketPath: "/ws/updates/process"
processInstanceUid: "cb19bfc0-cacb-4cbe-96bc-73c5771c0699"
webSocketAddress: "wss://public.qa.flowxai.dev/cb19bfc0-cacb-4cbe-96bc-73c5771c0699"
```

- **Status** - status of the process instance, possible values:
 - CREATED - the status is visible if there is an error in the process creation. If there is no error, the "Started" status is displayed.
 - STARTED - indicates that the process is currently running
 - DISMISSED - the status is available for processes with subprocesses, it is displayed when a user stops a subprocess
 - EXPIRED - the status is displayed when the "expiryTime" field is defined in the process definition and the defined time has passed.
 - FINISHED - the process has successfully completed its execution
- **Process definition** - the name of the process definition
- **Active process instance** - the UUID of the process instance, with a copy action available

- **Variables** - displayed as an expanded JSON

Process definition: **Awesome Process** Active process instance: **12156183...** 

Variables Tokens Subprocesses Exceptions **10**

```
processInstanceId: 619702
tokenId: 619752
tokenUuid: "147aa0c4-d961-4154-8af6-3422e1d34fb6"
webSocketPath: "/ws/updates/process"
processInstanceUuid: "12156183-b40d-4cee-b0e0-e296371cedbf"
webSocketAddress: "wss://public.qa.flowxai.dev/12156183-b40d-4cee-b0e0-e296371cedbf"
```

- **Tokens** - a token represents the state within the process instance and describe the current position in the process flow

Process definition: **Awesome Process** Active process instance: **12156183...** 

Variables **Tokens** Subprocesses Exceptions **10**

Token uuid	Token Status	Status Current Node	Date updated	
147aa0c4-d961-4154-8af6-3422e1d34fb6	ACTIVE	EXECUTED_COMPLETE	09 Aug 2022, 4:06 PM	 

(!) INFO

For more information about token status details, here.

- **Subprocesses** - **!** displayed only if the current The fallback content to display on prerendering generated a **subprocess** instance

- **Exceptions** - errors that let you know where the process is blocked, with a direct link to the node where the process is breaking for easy editing

Process definition: **Awesome Process** Active process instance: **12156183...**  

Variables	Tokens	Subprocesses	Exceptions 10
Source	Message	Type	Timestamp
NODE (start Node)	Outgoing node for process instance not fo...	Node Definition	09 Aug 2022 at 4:06 F  
NODE (start Node)	Outgoing node for process instance not fo...	Node Definition	09 Aug 2022 at 4:06 F  
NODE (start Node)	Outgoing node for process instance not fo...	Node Definition	09 Aug 2022 at 4:06 F  

⚠ INFO

For more information on token status details and exceptions, check the following section:

» Failed process start

- **Audit Log** - the audit log displays events registered for process instances, tokens, tasks, and exceptions in reverse chronological order by timestamp

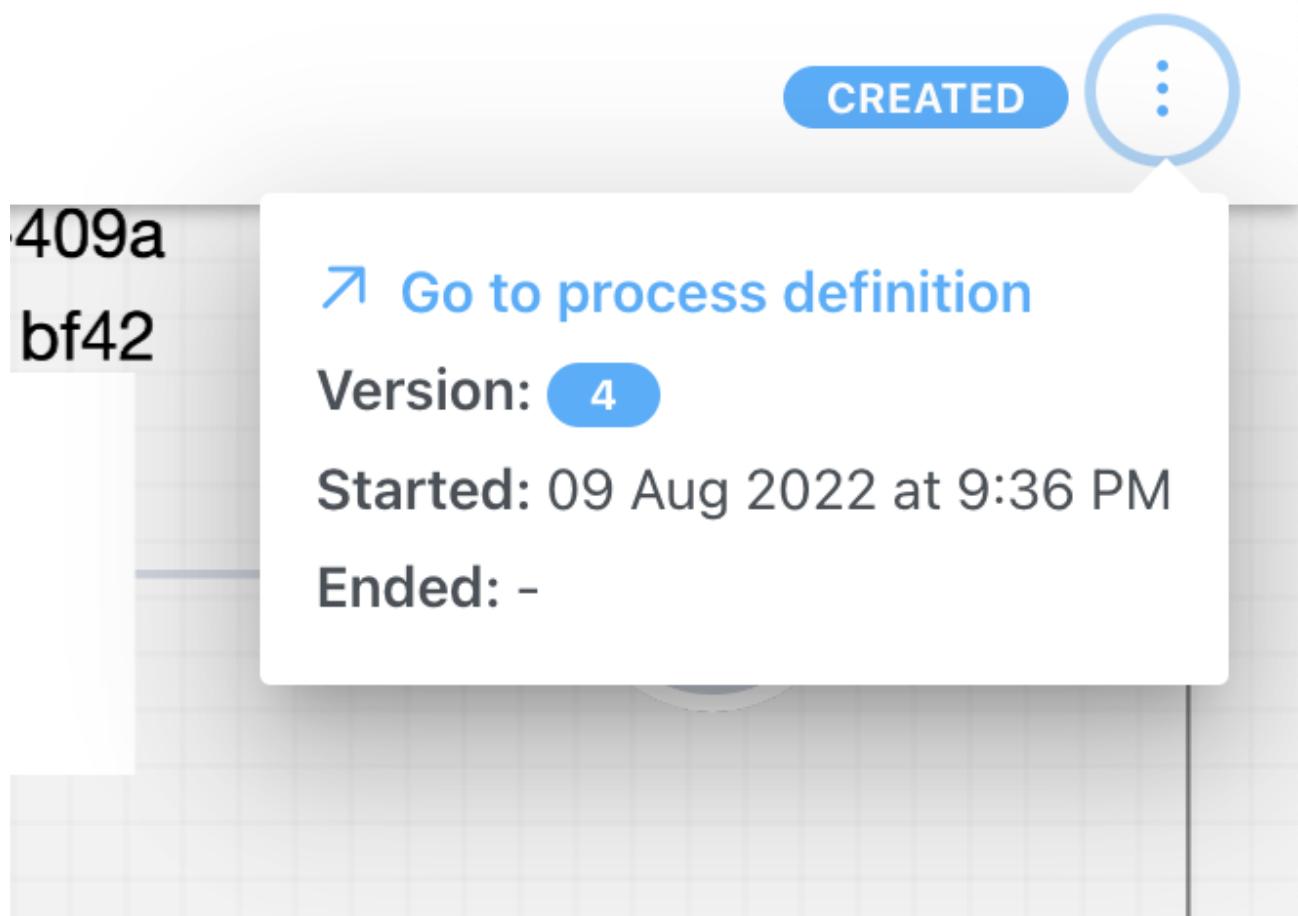
Variables	Tokens	Subprocesses	Audit Log	Exceptions 1	
Timestamp	User	Subject	Event	Subject Identifier	Status
06 Oct 2022, 1:54 PM	john.doe@email.com	Exception	View	-	- 
06 Oct 2022, 1:54 PM	system	Exception	Created	714951	- 
06 Oct 2022, 1:54 PM	john.doe@email.com	Process Instance	Start	a310c7b5-fb1c-4cf9-9b61-6badd174	success 

» Audit

Process menu

In the breadcrumb menu (top-right corner), you can access the following:

- **Go to process definition** - opens the process for editing
- **Version** - version of the process definition
- **Started** - timestamp for when the process instance started
- **Ended** - timestamp for when the process instance ended



Color coding

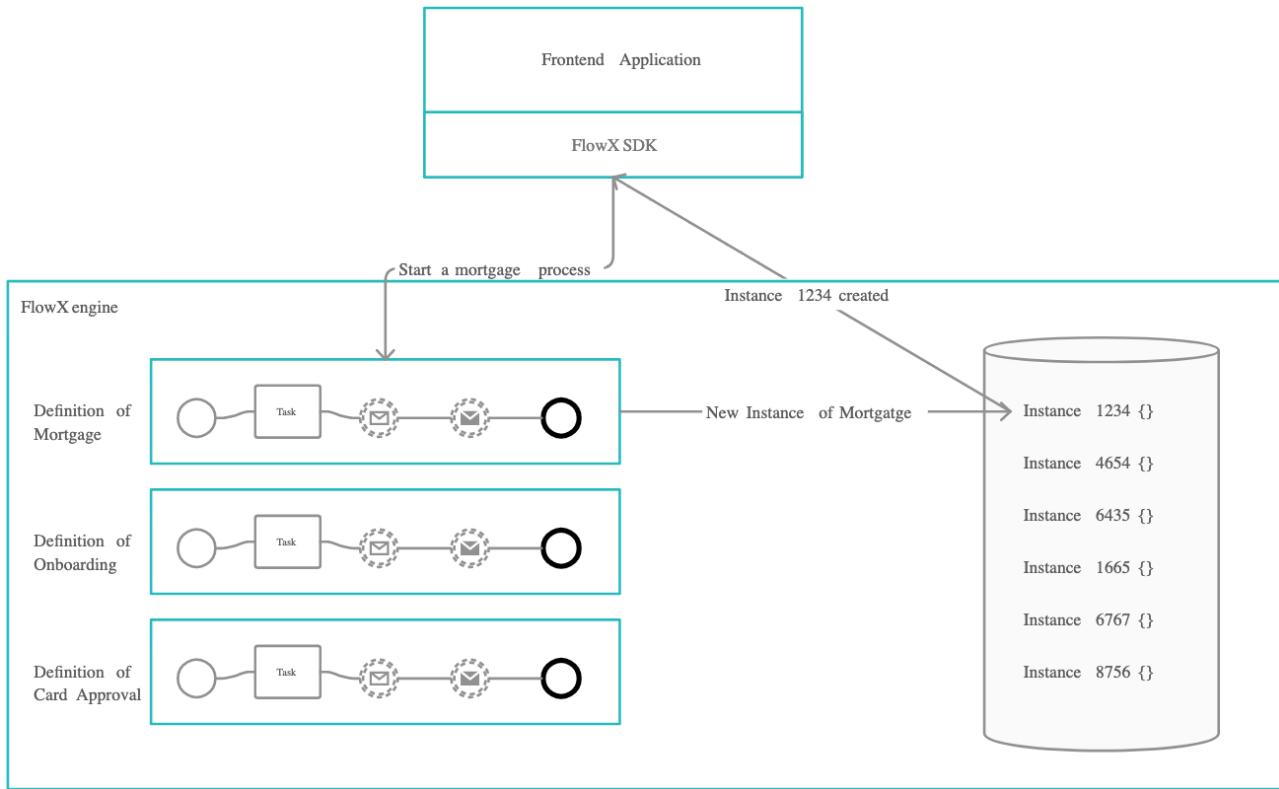
In the **Process Status** view, some nodes are highlighted with different colors to easily identify any failures:

- **Green** - nodes highlighted with green mark the nodes passed by the token
- **Red** - the node highlighted with red marks the node where the token is stuck (process failure)

Process uid	Definition name	Status	Current Node Name	Start date
a2f7ff1ef-ad39-4c00-8e58-492059f6e670	test_render_proc_instance	STARTED	Step	22 Jul 2022, 4:41 PM
e14bcfd3-a10e-4599-b0b4-e76669685fed	test_render_proc_instance	CREATED	Start	22 Jul 2022, 4:37 PM
037a6efe-0949-4f2b-ac8e-a94afe88d287	test_render_proc_instance	FINISHED	End	22 Jul 2022, 4:34 PM
0df92b59-6195-472a-bb7e-cb9cd98e0ad0	test_render_proc_instance	STARTED	Step	22 Jul 2022, 4:33 PM
2d403707-c480-48d0-a7ce-1d1603fa0ebc	test_render_proc_instance	STARTED	Step	22 Jul 2022, 4:31 PM
4fbe4e81-09a6-49d0-9f0b-aba566b896d8	test_render_proc_instance	CREATED	Step	22 Jul 2022, 4:04 PM

Starting a new process instance

To start a new process instance, a request must be made to the **FLOWX Engine**. This is handled by the web/mobile application. The current user must have the appropriate role/permission to start a new process instance.



To be able to start a new process instance, the current user needs to have the appropriate role/permissions:

» [Configuring access roles for processes](#)

When starting a new process instance, we can also set it to [inherit some values](#) from a previous process instance.

Troubleshooting possible errors

If everything is configured correctly, the new process instance should be visible in the UI and added to the database. However, if you encounter issues, here are

some common error messages and their possible solutions: Possible errors include:

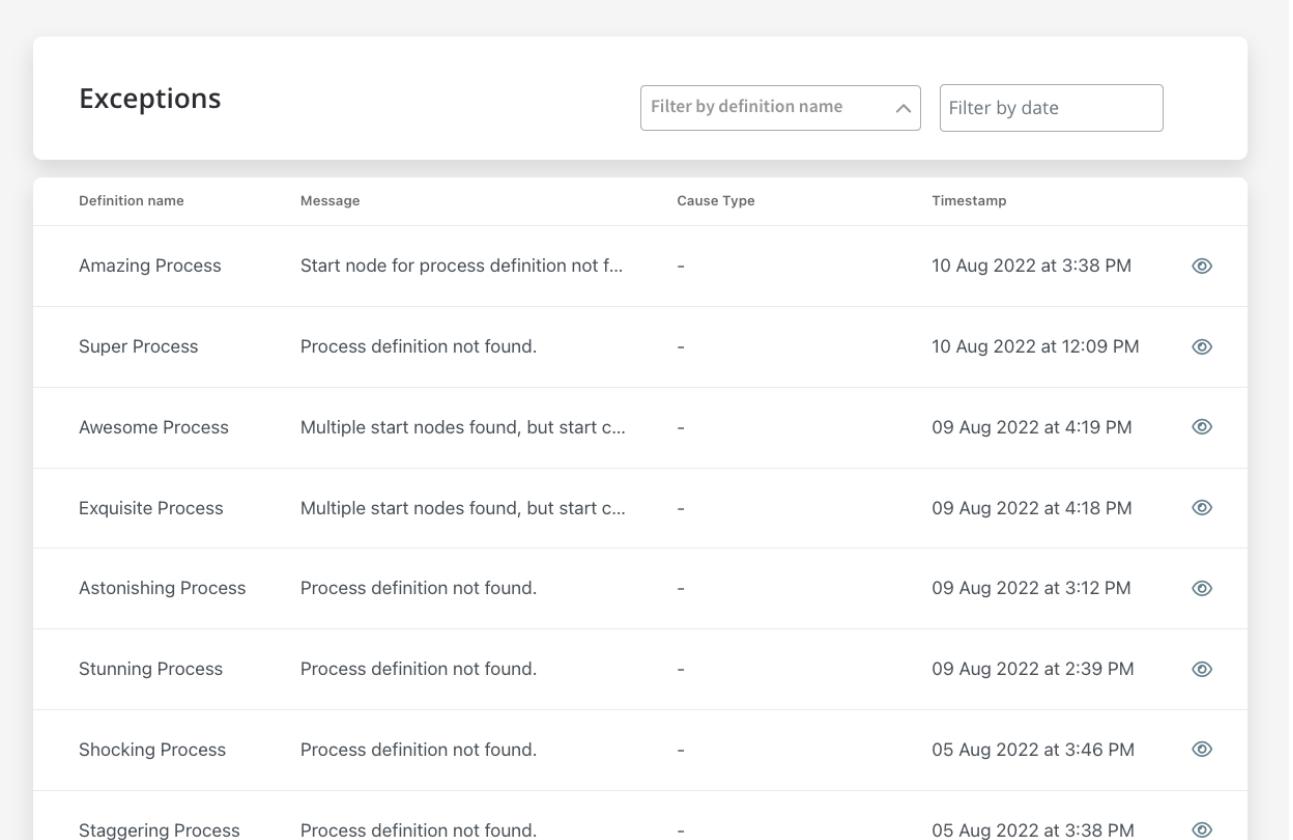
Error Message	Description
<i>"Process definition not found."</i>	The process definition with the requested name was not set as published.
<i>"Start node for process definition not found."</i>	The start node was not properly configured.
<i>"Multiple start nodes found, but start condition not specified."</i>	Multiple start nodes were defined, but the start condition to choose the start node was not set.
<i>"Some mandatory params are missing."</i>	Some parameters set as mandatory were not included in the start request.
HTTP code 403 – Forbidden	The current user does not have the process access role for starting that process.
HTTP code 401 – Unauthorized	The current user is not logged in.

Was this page helpful?

BUILDING BLOCKS / Process Designer / Active process / Failed process start

Exceptions

Exceptions are types of errors meant to help you debug a failure in the execution of a process.



The screenshot shows a table titled 'Exceptions' with columns: Definition name, Message, Cause Type, and Timestamp. There are two filter buttons at the top right: 'Filter by definition name' and 'Filter by date'. The table lists nine entries, each with a small circular icon next to the timestamp.

Definition name	Message	Cause Type	Timestamp	
Amazing Process	Start node for process definition not f...	-	10 Aug 2022 at 3:38 PM	🔗
Super Process	Process definition not found.	-	10 Aug 2022 at 12:09 PM	🔗
Awesome Process	Multiple start nodes found, but start c...	-	09 Aug 2022 at 4:19 PM	🔗
Exquisite Process	Multiple start nodes found, but start c...	-	09 Aug 2022 at 4:18 PM	🔗
Astonishing Process	Process definition not found.	-	09 Aug 2022 at 3:12 PM	🔗
Stunning Process	Process definition not found.	-	09 Aug 2022 at 2:39 PM	🔗
Shocking Process	Process definition not found.	-	05 Aug 2022 at 3:46 PM	🔗
Staggering Process	Process definition not found.	-	05 Aug 2022 at 3:38 PM	🔗

Exceptions can be accessed from multiple places:

- **Failed process start** tab from **Active process** menu in FLOWX Designer

- **Process Status** view, accessible from **Process instances** list in FLOWX Designer

Process definition: **Awesome Process** Active process instance: **12156183...** □

Variables Tokens Subprocesses **Exceptions** 10

Source	Message	Type	Timestamp	Actions
NODE (start Node)	Outgoing node for process instance not fo...	Node Definition	09 Aug 2022 at 4:06 F	ⓘ ⓧ
NODE (start Node)	Outgoing node for process instance not fo...	Node Definition	09 Aug 2022 at 4:06 F	ⓘ ⓧ
NODE (start Node)	Outgoing node for process instance not fo...	Node Definition	09 Aug 2022 at 4:06 F	ⓘ ⓧ

⚠ CAUTION

If you open a process instance and it does not contain exceptions, the **Exceptions** tab will not be displayed.

Exceptions data

When you click **view** button, a detailed exception will be displayed.

Exceptions: 619820

Process Definition:	Awesome Process	Cause Type:	START_EVENT
Source:	NODE (start Node)	Process Instance UUID:	12156183-b40d-4cee-b0e0-e296371cedbf
Message:	Outgoing node for process instance not found.	Token UUID:	147aa0c4-d961-4154-8af6-3422e1d34fb6
Type:	Node Definition	Timestamp:	09 Aug 2022 at 4:06 PM

Details

```
1 [ai.flowx.enginedefinitions.dto.NodeDTOWrapper.lambda$getOutgoingNodeMandatory$4(NodeDTOWrapper.java:45),  
java.base/java.util.Optional.orElseThrow(Optional.java:408), ai.flowx.enginedefinitions.dto.NodeDTOWrapper.  
getOutgoingNodeMandatory(NodeDTOWrapper.java:45), ai.flowx.engine.instance.service.impl.  
NodeProcessorServiceImpl.getNextNodeId(NodeProcessorServiceImpl.java:249), ai.flowx.engine.instance.service.  
impl.NodeProcessorServiceImpl.process(NodeProcessorServiceImpl.java:155), ai.flowx.engine.instance.service.  
impl.NodeProcessorServiceImpl$$FastClassBySpringCGLIB$$badf4a43.invoke(<generated>), org.springframework.  
cglib.proxy.MethodProxy.invoke(MethodProxy.java:218), org.springframework.aop.framework.  
CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:779), org.springframework.aop.  
framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163), org.springframework.aop.  
framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:750), org.springframework.aop.  
aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:89), ai.flowx.  
commons.trace.aop.JaegerTraceAspect.around(JaegerTraceAspect.java:52), jdk.internal.reflect.  
GeneratedMethodAccessor582.invoke(Unknown Source), java.base/jdk.internal.reflect.  
DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43), java.base/java.lang.reflect.Method.  
invoke(Method.java:566), org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs  
(AbstractAspectJAdvice.java:634), org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod  
(AbstractAspectJAdvice.java:624), org.springframework.aop.aspectj.AspectJAroundAdvice.invoke  
(AspectJAroundAdvice.java:72), org.springframework.aop.framework.ReflectiveMethodInvocation.proceed  
(ReflectiveMethodInvocation.java:175), org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.  
proceed(CglibAopProxy.java:750)]
```

- **Process Definition** - the process where the exception was thrown
- **Source** - the source of the exception (see the possible type of **sources** below)
- **Message** - a hint type of message to help you understand what's wrong with your process

- **Type** - exception type
- **Cause Type** - cause type (or the name of the node)
- **Process Instance UUID** - process instance unique identifier
- **Token UUID** - token unique identifier
- **Timestamp** - default format: yyyy-MM-dd 'T'HH:mm:ss.SSSZ
- **Details** - stack trace (a **stack trace** is a list of the method calls that the process was in the middle of when an **Exception** was thrown)

Possible sources:

- Action
- Node
- Subprocess
- Process Definition

Exceptions type

Based on the exception type, there are multiple causes that could make a process fail. Here are some examples:

Type	Cause
Business Rule Evaluation	when executing action rules fails for any reason
Condition Evaluation	when executing action conditions

Type	Cause
Engine	<p>when the connection with the database fails</p> <p>when the connection with Redis fails</p>
Definition	misconfigurations: process def name, subprocess parent process id value, start node condition missing
Node	when an outgoing node can't be found (missing sequence etc)
Gateway Evaluation	<p>when the token can't pass a gateway for any reason, possible causes:</p> <ul style="list-style-type: none"> • missing sequence/node • failed node rule
Subprocess	exceptions will be saved for them just like for any other process, parent process ID will also be saved (we can use this to link them when displaying exceptions)

Was this page helpful?

BUILDING BLOCKS / Process Designer / Subprocess

Sub-processes are smaller process flows that can be triggered by actions in the main process. They can also inherit some process parameter values from the parent process and send their results back to the parent process when they are completed. The subprocesses will communicate with the front-end apps using the same connection details as their parent process.

They can be started in two ways:

- **asynchronous** - they will execute alongside the parent process since the parent process does not need to wait for the sub-process to end
- **synchronous** - the parent process will wait until the sub-processes are finished before advancing

Configuring & starting subprocesses

The sub-processes will be designed in the same way as the main process, by using the FLOWX Designer.

They can be started by a parent process in one of two ways:

- by using a StartSubprocess action inside any of the task nodes in the process
- by adding a custom Subprocess Run node type in the process

In both cases, by default, the sub-process will inherit all the parent process parameter values. It can be configured to inherit only some parameter values from

its parent. The available action parameters for this are:

- *paramsToCopy* - choose which of the keys from the parent process parameters to be copied to the sub-process
- *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 sub-process

Sub-processes can have an action configured on them which will append their results to the parent process parameter values.

Executing subprocesses

The sub-processes can be started in async or sync mode, by setting a specific action parameter, named *startedAsync*, on the action that triggers the subprocess.

If the subprocesses are started in sync mode, they will notify the parent process when they are completed and the parent process will handle receiving the process data from the child and resuming its flow.

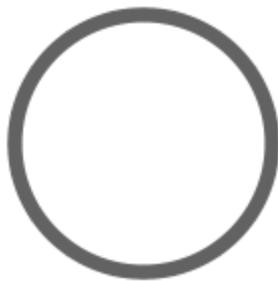
[Was this page helpful?](#)

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.

The screenshot shows the FLOWX.AI process builder interface. At the top, there's a toolbar with icons for settings, user, plus, and minus. A search bar is labeled "Search by node name". Below the toolbar, a title "Start Node Example" is displayed next to a small blue outline of a circle. On the left, there's a sidebar with a "Node Config" tab selected, showing "General Config" with a "Node name" field containing "Start Node Example". There's also a toggle switch for "Can go back?". Under "Swimlane", a dropdown menu is set to "Default". At the bottom right, there's a "Save" button.

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

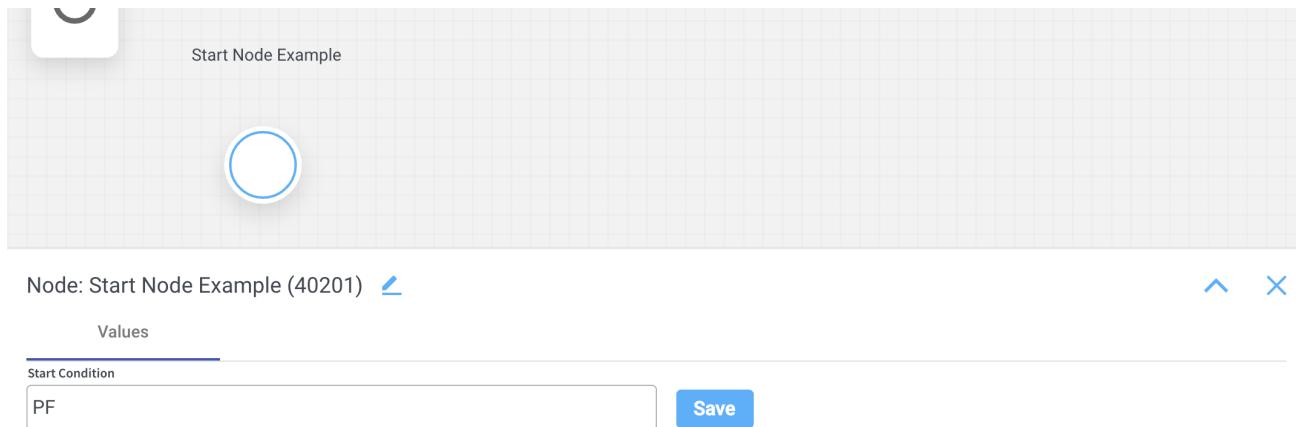
(!) INFO

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.NodeDefinition",  
    "defaultMessage": "Start node for process definition not found",  
    "errorKey": "error.validation.process_instance.start_node_for_process_definition_not_found",  
    "type": "https://www.jhipster.tech/problem/problem-with-message",  
    "title": "Start node for process definition not found.",  
    "status": 400,  
    "message": "error.validation.process_instance.start_node_for_process_definition_not_found",  
    "params": "ai.flowx.process.definition.domain.NodeDefinition",  
}
```

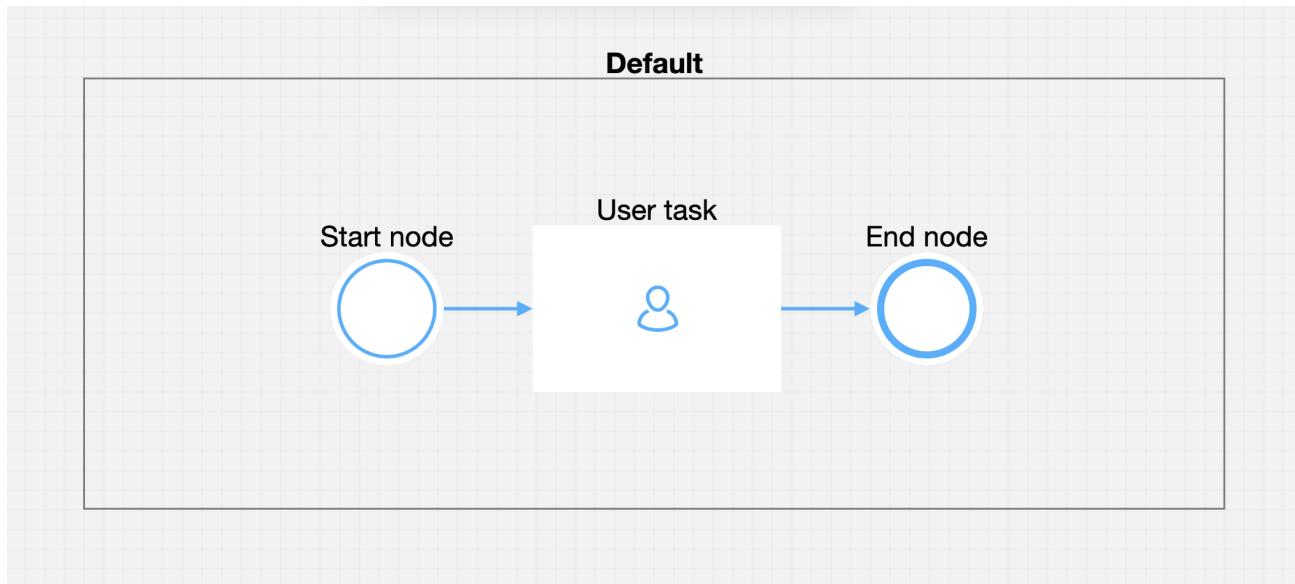
End node



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

Node: **Message send task** (ID: 546054)

Node Config Actions

General Config

Node name

Message send task

Can go back?

Swimlane

Default ▼

Stage

▼

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

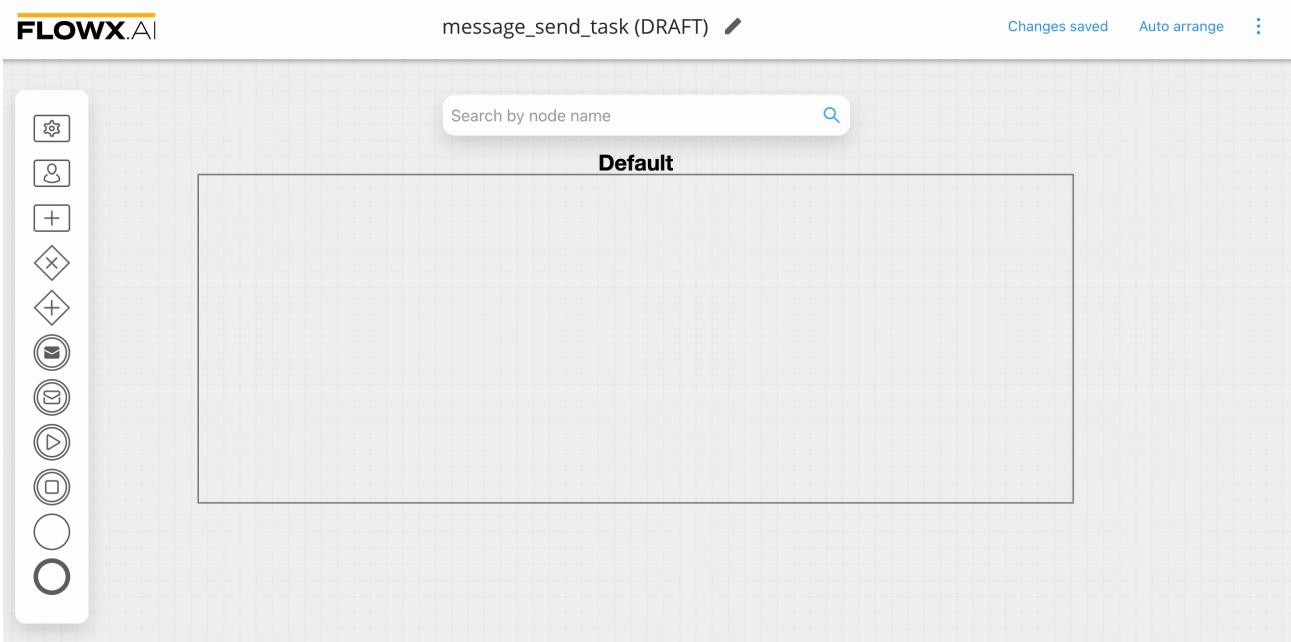
(**Kafka Send Action** type):

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

Action Edit

ID: 540663

Name

f29bcac4-4e42-4e0d-9b66-953d67f272c4

Order

1

Timer Expression

Kafka Send Action



Automatic Manual

Mandatory Optional

Repeatable

Autorun Children?

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

Address

Replace Values

Message

1

Save

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}}`

Action Edit

ID: 540663

Name

sendRequestToSearchClient

Order

1

Timer Expression

Kafka Send Action



Automatic Manual

Mandatory Optional

Repeatable

Parameters

Custom

From integration

Topics

```
ai.flowx.in.crm.search.v1
```

Message

```
1 {  
2   "clientType": "${application.client.clientType}",  
3   "personalNumber": "${personalNumber.client.personalNumber}"  
4 }
```

Advanced configuration

Show Headers

Save

Advanced configuration

Show Headers

```
1 {"processInstanceId": ${processInstanceId}}
```

Replace Values

Save

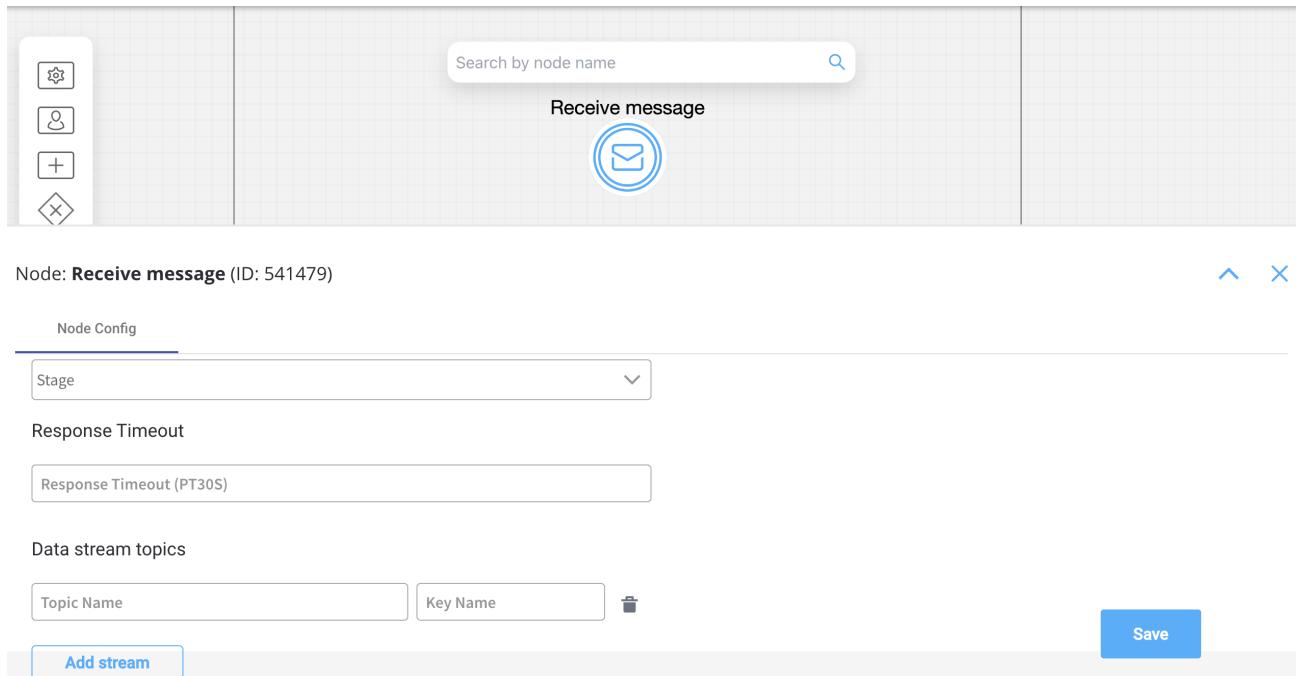
Message receive task

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`

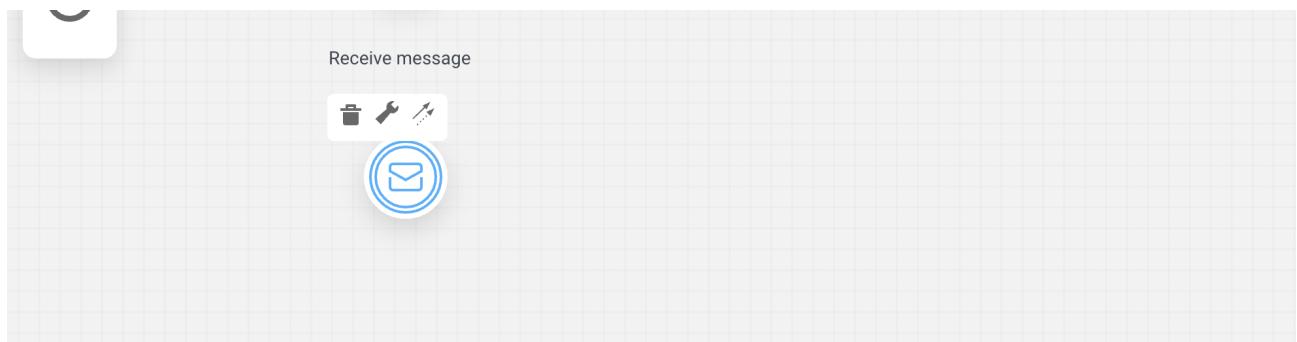
🔥 DANGER

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](#).



Node: Receive message (40221) [🔗](#)

Values

Response Timeout (PT30S)		Save
Topic Name	Key Name	
ai.flowx.out.crm.search.v1	crmResponse	Delete
Topic Name	Key Name	
		Add

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

1

Timer Expression

Kafka Send Action ▾

Automatic Manual

Mandatory Optional

Repeatable

Autorun Children?

Parameters 

Custom From integration

Select a scenario ▾

Save

Was this page helpful?

BUILDING BLOCKS / Node / Task node

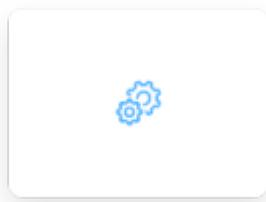
A task

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

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

Node Config	Actions
<p>General Config</p> <p>Node name</p> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;">Task node</div> <p>Can go back? <input checked="" type="checkbox"/></p>	

! **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**)

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

Node Config	Actions
Swimlane	<input type="text" value="Default"/> 
Stage	<input type="text" value=""/> 
Response Timeout	<input type="text" value="Response Timeout (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

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

Node Config

Actions

Data stream topics

Topic Name

Key Name



Add stream

Task Management

Update task management?

i Force Task Management 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

Action Edit

ID: 31808

Name

action75

Order

1

Timer Expression

Business Rule



Automatic



Manual



Mandatory



Optional



Repeatable

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

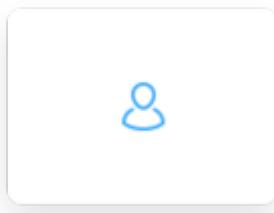
Was this page helpful?

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

Node: **User task** (ID: 545152)

Node Config

Actions

General Config

Node name

User task

Can go back?

Flow Names

Leave empty if this node is to 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`)

Node: **User task** (ID: 545152)

Node Config

Actions

Swimlane

Default



Stage



Response Timeout

Response Timeout (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

Node: **User task** (ID: 545152)

Node Config	Actions
Data stream topics	
Topic Name	Key Name 
Add stream	

Task Management

Update task management? 

 Force Task Management Plugin 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**.

The screenshot shows the FLOWX.AI interface with a sidebar on the left containing navigation links for Processes, Content Management, Plugins, and Task Manager. The main area is titled 'Process Definitions' and contains two sections: 'Drafts / In progress' and 'Published'. The 'Drafts / In progress' section lists 'Test Process' (version 1, edited on 06 Jun 2022 at 4:33 PM by John Doe) and 'Test Process 2' (version 2, edited on 06 Jun 2022 at 9:53 AM by Jane Doe). The 'Published' section lists 'Happy Process' (version 4, published on 26 May 2022 at 12:49 PM by John Doe) and 'Stepper Process' (version 1, published on 24 May 2022 at 12:13 PM by Jane Doe). A search bar and a more options button are visible at the top right of the main content area.

» 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 multiple properties that can be managed.

1. **Input** - FLOWX form element that allows you to generate an input form field
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-04df0815fb0",
"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?](#)

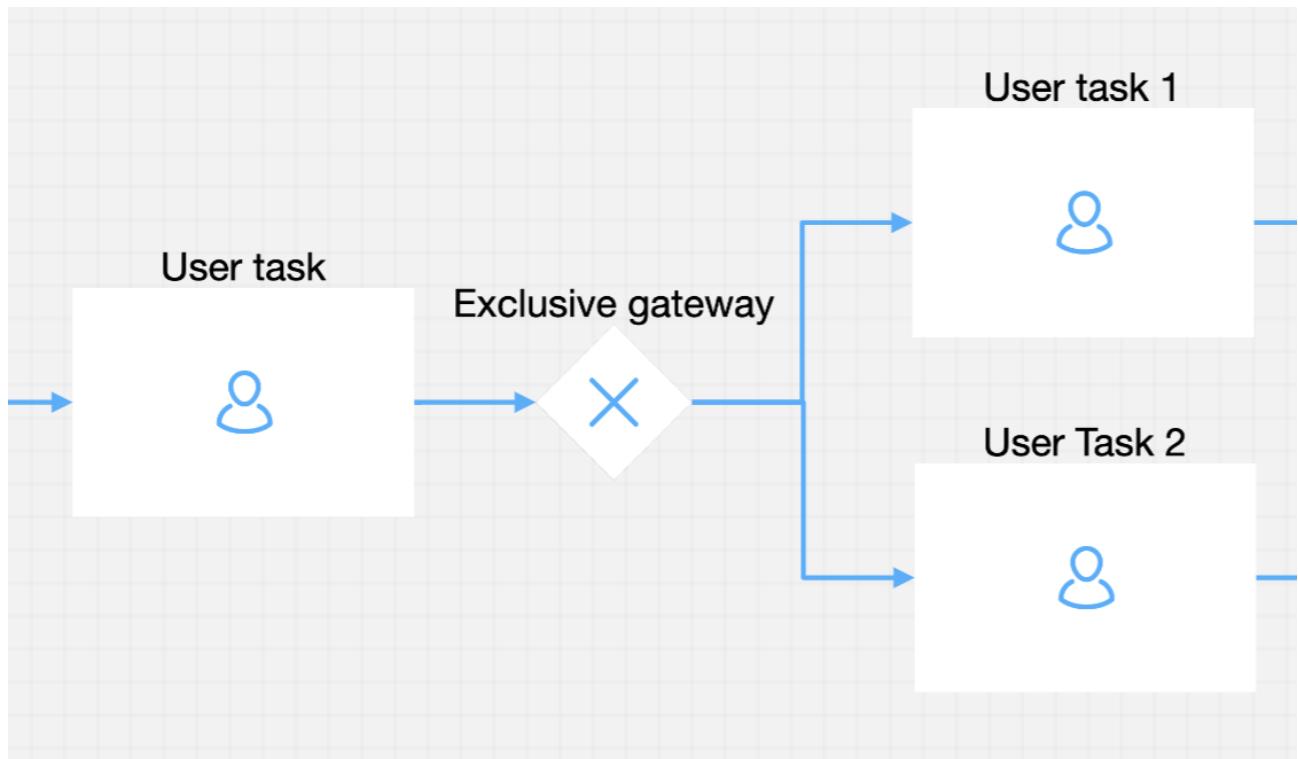
BUILDING BLOCKS / Node / Exclusive gateway

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

Node: **12f5bcc0-13f3-44d8-b9c2-4e00c143fff4** (ID: 545174) ▼ X

Node Config Gateway Decisions

General Config
Node name
exclusive gateway

Can go back?

Swimlane
Default

Stage

Save

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.

Node: **Exclusive gateway** (ID: 501853) ^

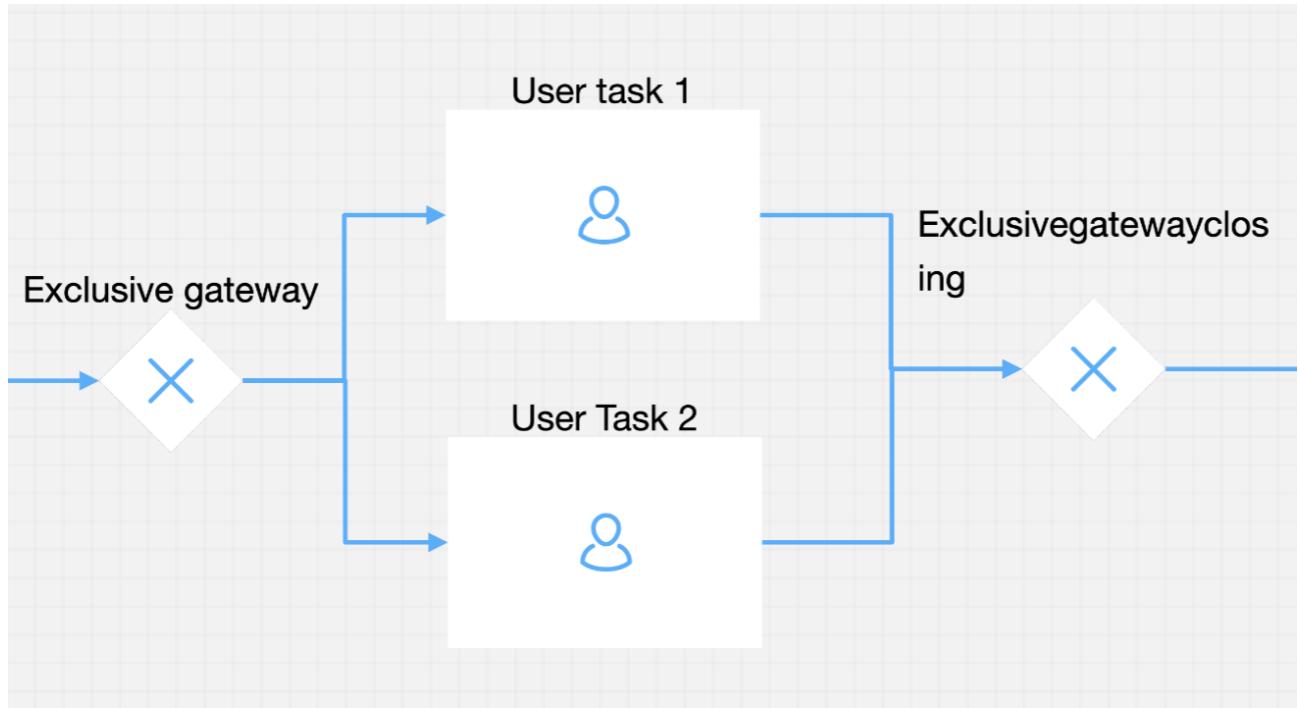
Node Config Gateway Decisions

Language: MVEL

if	↳ (input.get("application.client.age") < 18) then go to	User task 1	✖
else if	↳ (input.get("application.client.age") < 18) then go to	User task 1	✖
else if	↳ (Expression) then go to	⋮	Add

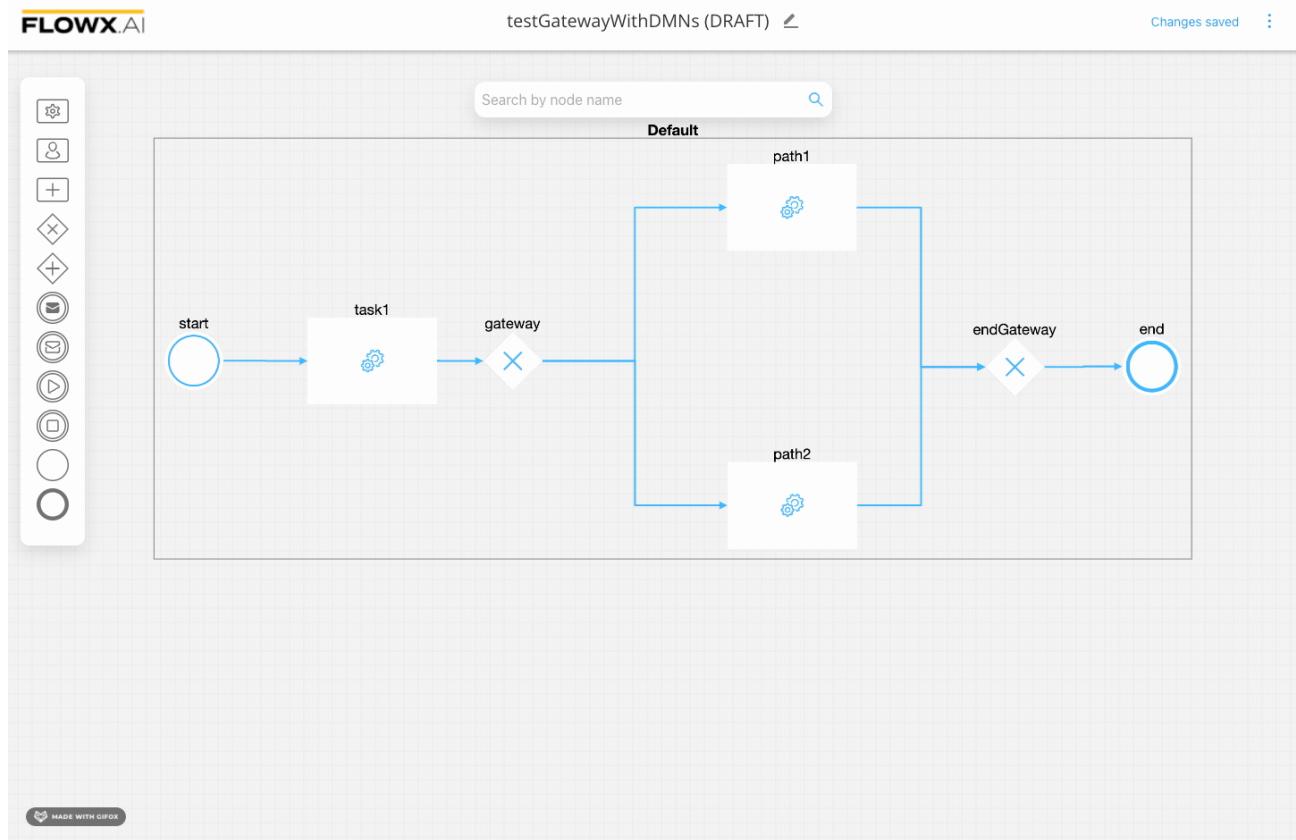
Save

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)

Node: **gateway** (ID: 367901)

[Node Config](#) [Gateway Decisions](#)

[Language](#)

Language

DMN

Rules

[View DRD](#)

		Hit Policy: First			
	When	Input	Then	Next Node Name	Annotations
		boolean		string	
1	true			path1	
2	false			path2	
+	-				

[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.

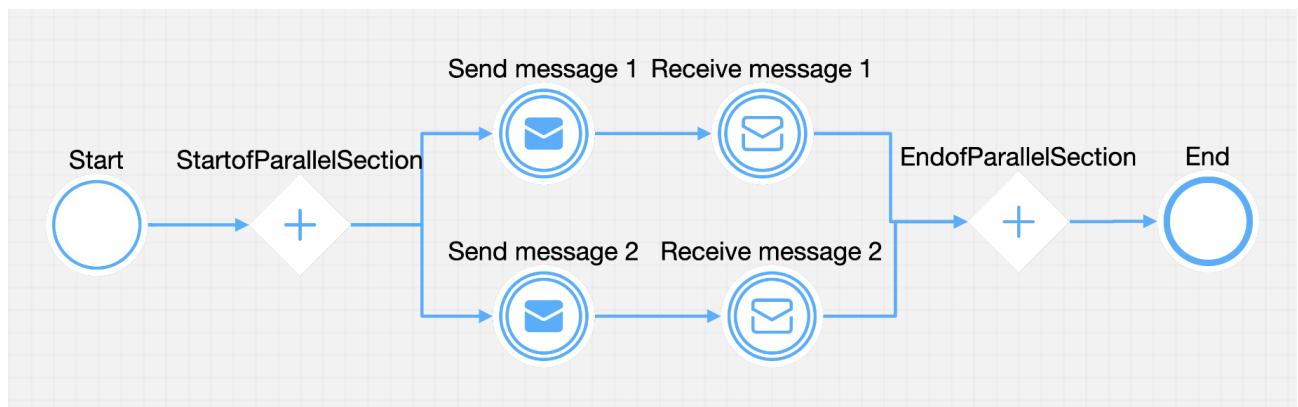
Configuring a Parallel gateway node



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:

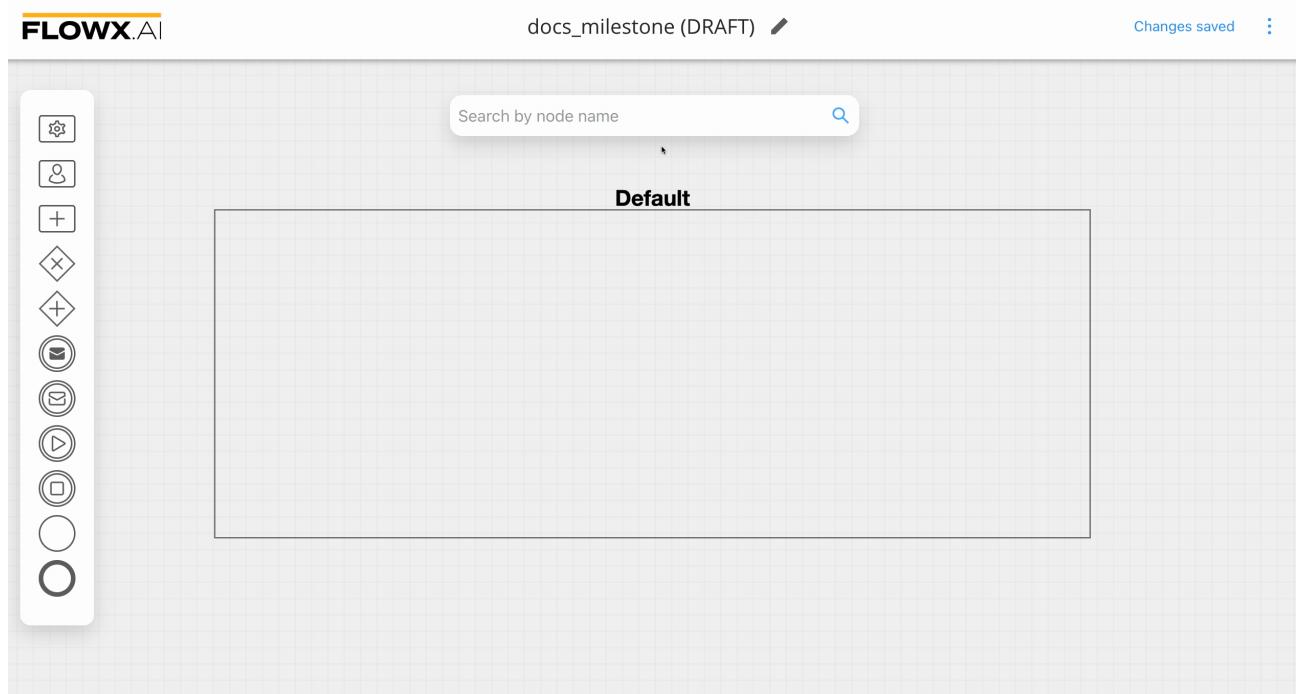
- **Modal**
- **Page**
- **Stepper + Steps**
- **Container**

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.

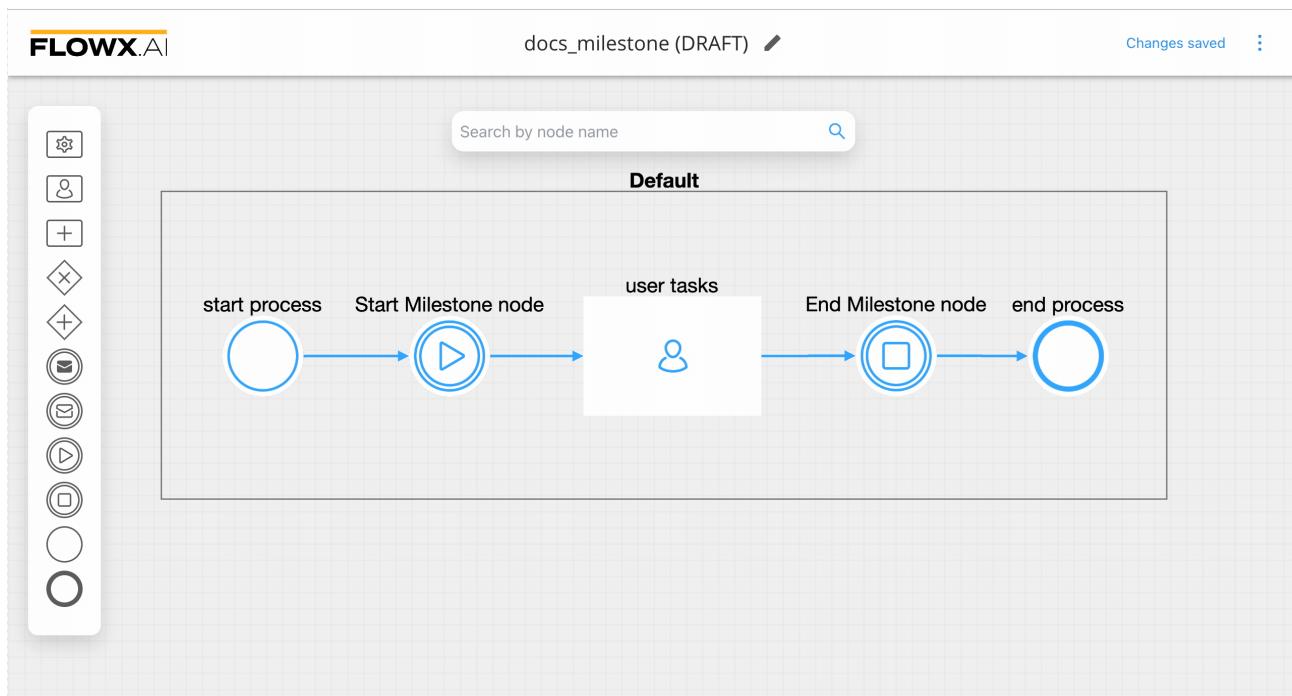
To configure a user task to be displayed in a Modal:

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.

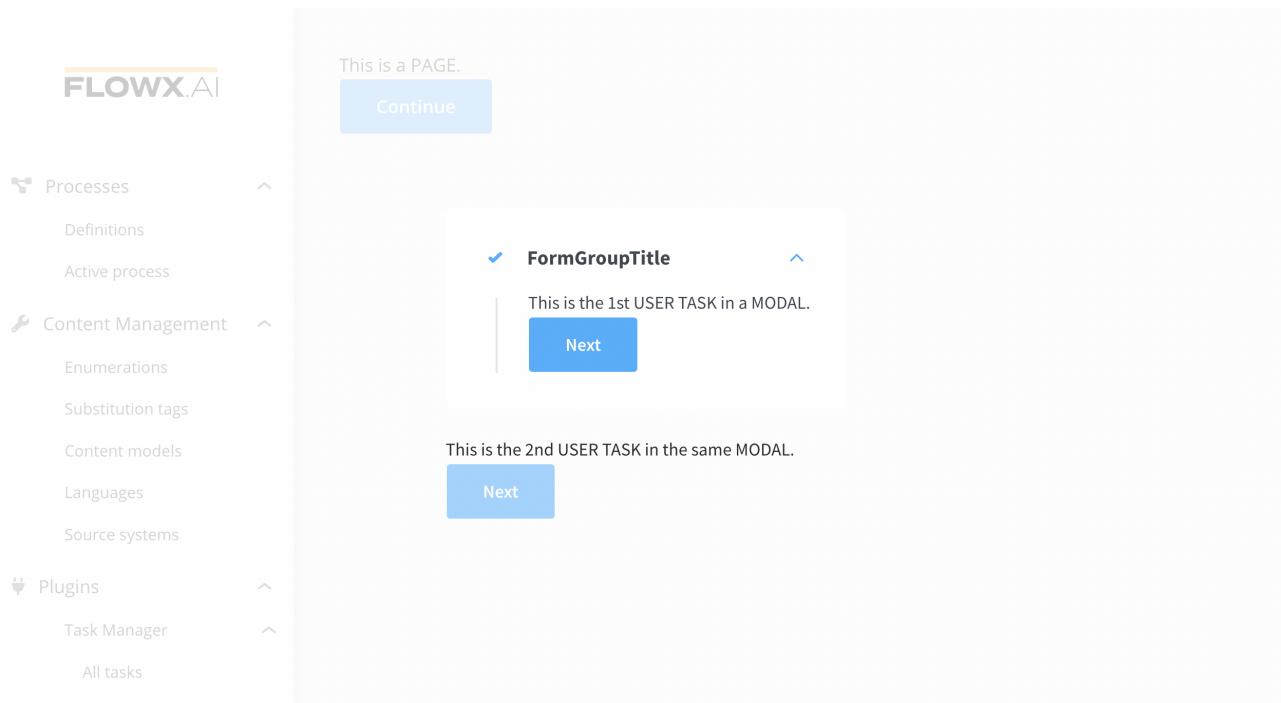
7. No additional information is required for displaying a **user task** in a modal view but you can do multiple customizations via the different configurations using the UI Designer.



Available Components

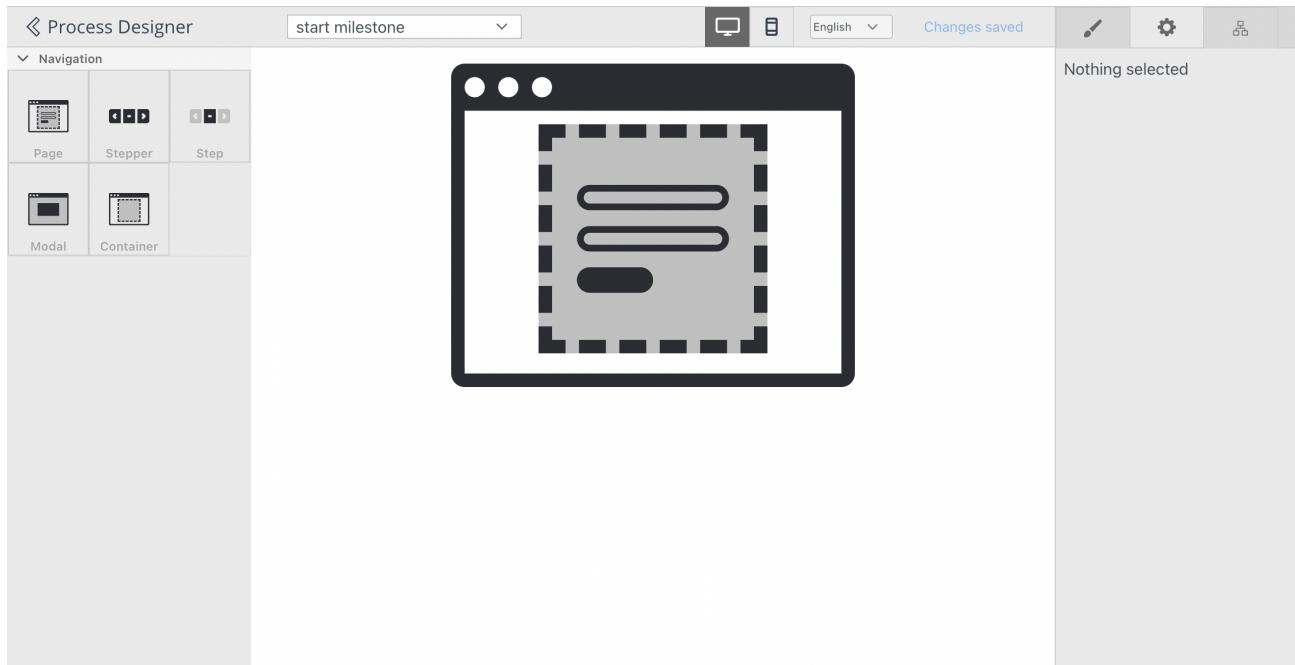
Modal

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

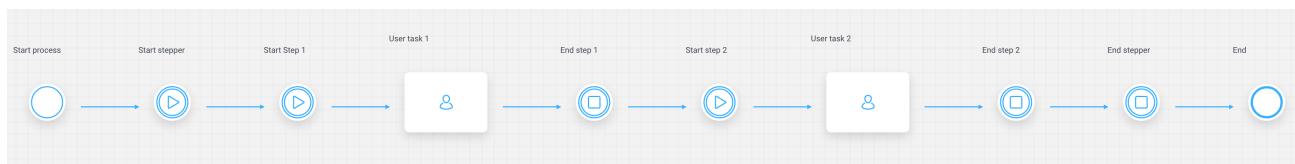
A basic full page content can be displayed using this kind of template on a milestone start.



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**.

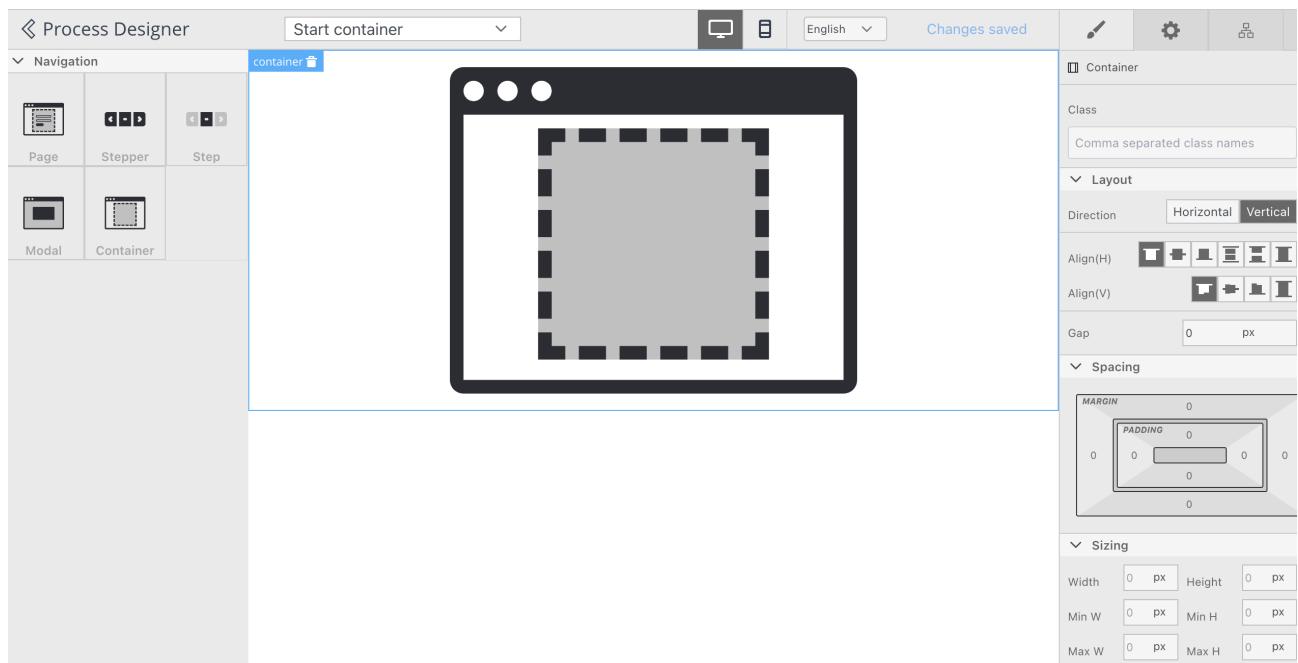


For more information about how to create a process with a Stepper + Steps structure, and how to configure the UI, check the following section:

» 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

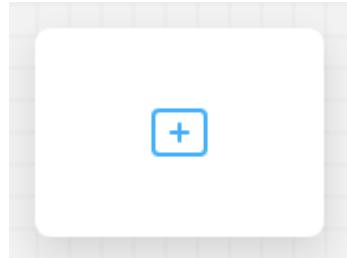
!(INFO)

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

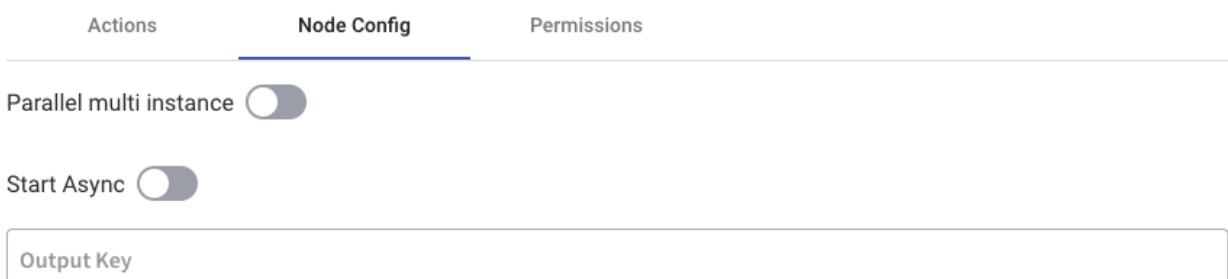
!(INFO)

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.

Actions Node Config Permissions

Parallel multi instance

Collection Key

Start Async

Output Key

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 the 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

Node: Application decision (ID: 2307723) ▼ ▲

Node Config

General Config

Can go back?

Correlate with catch events

branch1



Correlation Key

app.id

```
1 {"approved": "${app.approved}"}
```

Stage

Onboarding



[Save](#)

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

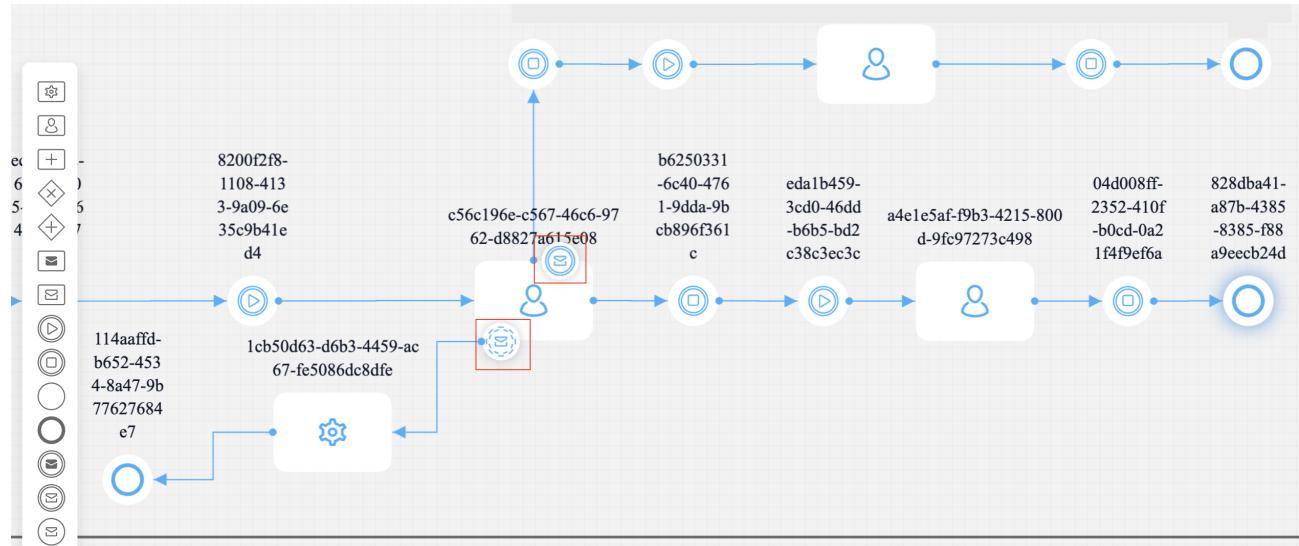
Why it is important? It allows the process to listen for and capture specific messages during the execution of the associated user task.

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

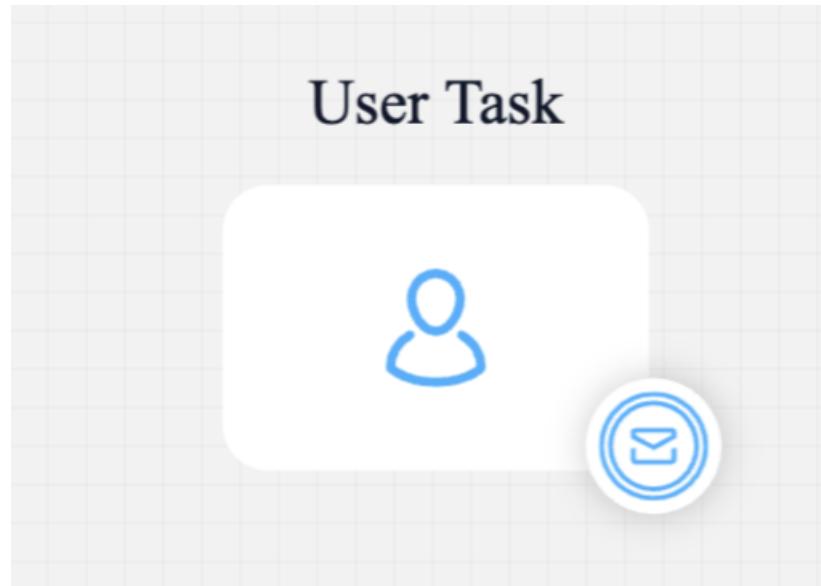
. You can have multiple intermediate boundary events on the same user task, but only one can be activated at a time.



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.

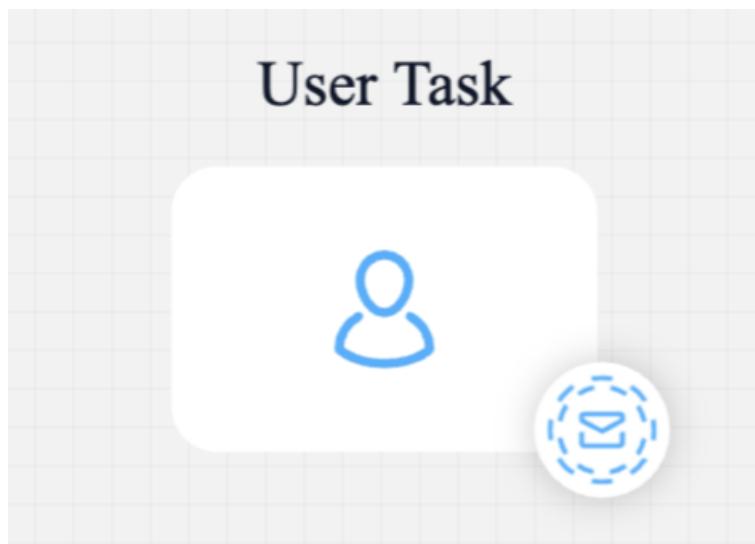
It can also be used as a standalone node, see more information on the following section:

» [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

Node: **ee61fcc9-aa51-4b8a-a4f1-ee2df56a23d6** (ID: 2313991)

Node Config

General Config

Can go back?

Correlate with throwing events

same_process



Correlation Key

app.id

Receive data

Process Key

test

Stage



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

Node: **cc20eb60-c74f-4e37-9a09-d3e1154582f1** (ID: 2309403)

Node Config

General Config

Can go back?

Correlate with throwing events

new_event



Correlation Key

qwe

Receive data

Process Key

receivedData

Stage



Was this page helpful?

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		Hit Policy: Unique		
	When	And		Annotations
	application.client.clientType string	alert_title string	alert_description string	
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>
    <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)



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

Parameters

Message Type

Message

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

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**.

Node: Task node (ID: 542401) ▼ X

Node Config Actions

▼ Actions + |

{ "name": "\${application.client.firstName}" }

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": "currentNodeName"}
```

 Replace Values

Data to send

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

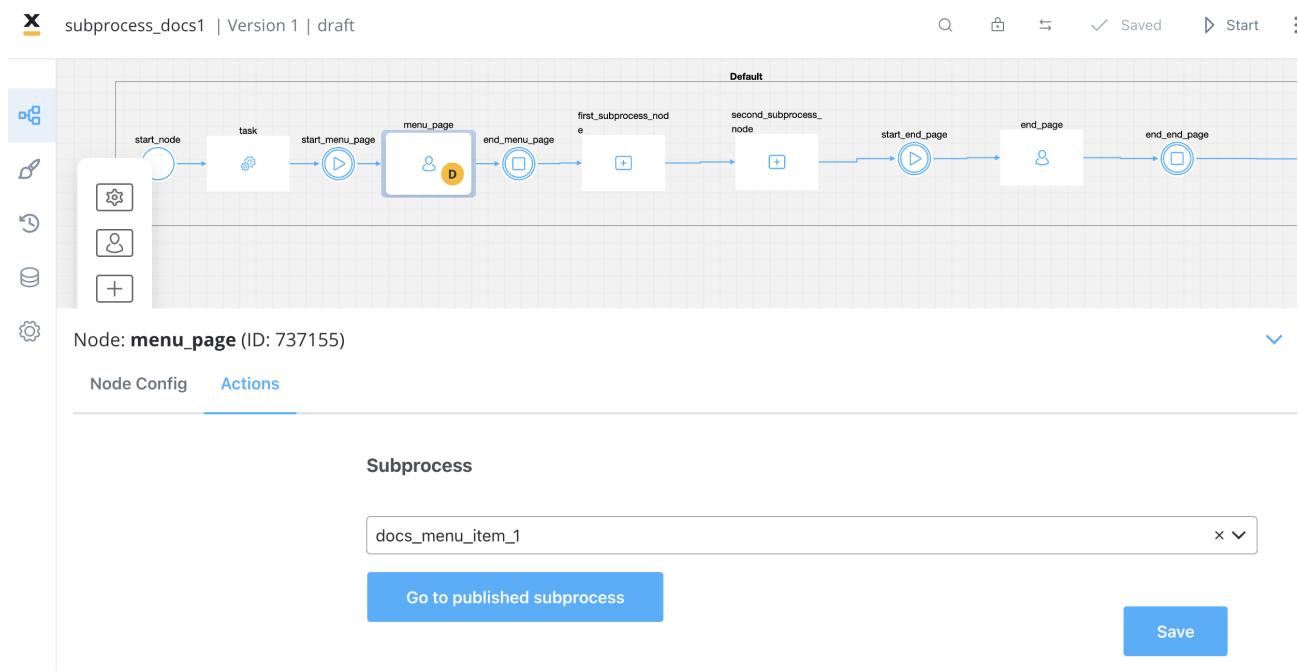
To use a process as a **subprocess**, you must first create it. Once the subprocess is created, you can start it from another (parent) process. To do this, you will need to add a **Start Subprocess** action to a **User task** node in the parent process or by using a **subprocess run node**.

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

selectedCustomer trash

selectedSubscription trash

url trash

trash

trash

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



Exclude from current state

Add Param

Copy from current state

Add Param

Advanced configuration

Show Target Process

Replace Values

Data to send

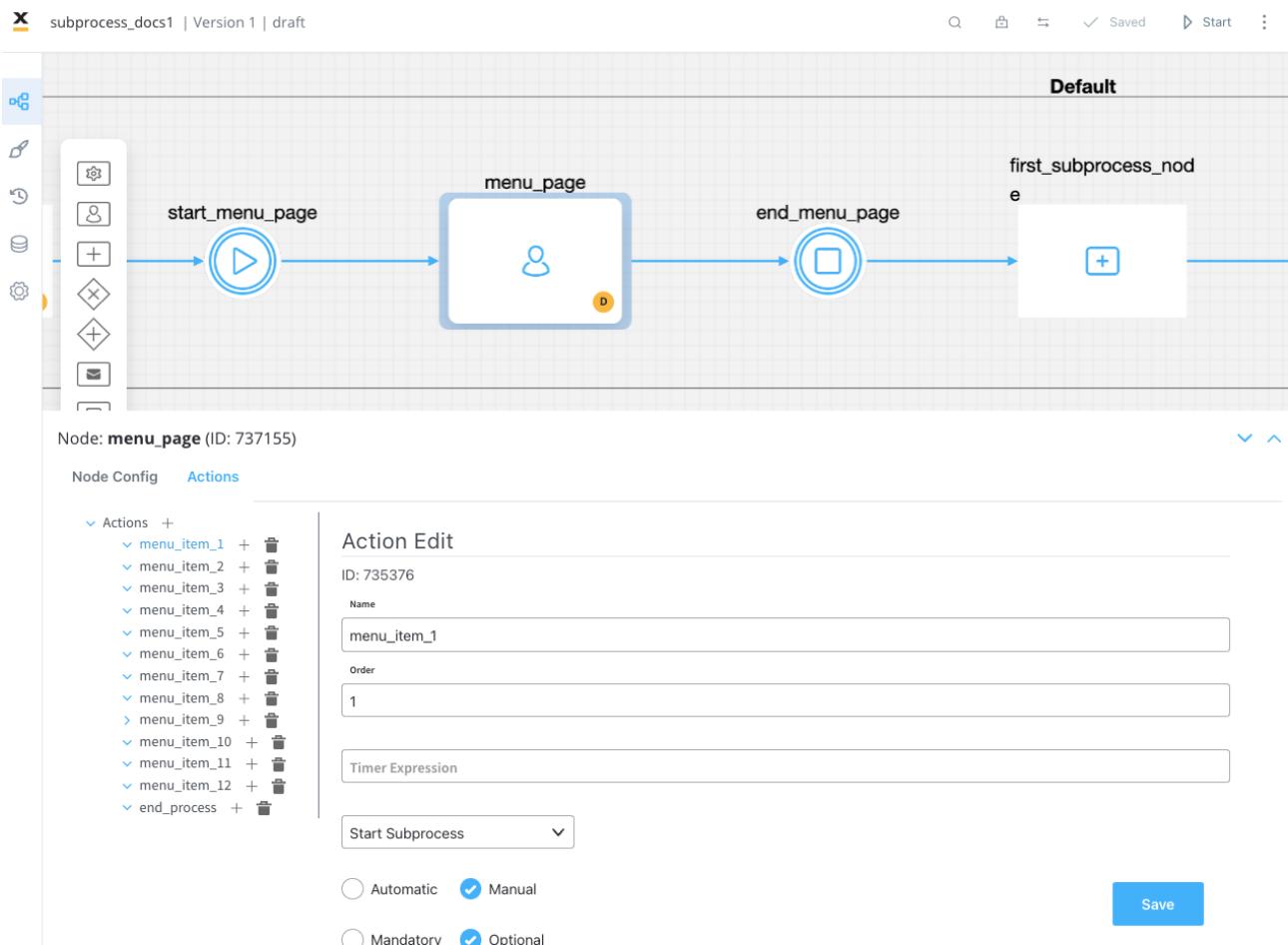
Add KeySave

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 configuration page for a subprocess named 'subprocess_docs1'. The top navigation bar includes a back arrow, a search icon, a save icon, a start icon, and a more options icon. The main area displays the node details: 'Node: menu_page (ID: 737155)' and tabs for 'Node Config' (selected) and 'Actions'. On the left, there's a sidebar with icons for edit, history, copy, and settings. The 'Parameters' section contains a single field 'Subprocess' with the value 'docs_menu_item_1'. Below it is a blue button labeled 'Go to published subprocess'. The 'Exclude from current state' section contains a field with 'test.price' and an 'Add Param' button. The overall interface is clean and modern, typical of a web-based configuration tool.

- **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

The screenshot shows the FLOWX.AI subprocess configuration interface for a subprocess named "menu_page". The "Actions" tab is selected. A "Copy from current state" section is present, listing several parameters with their current values:

- application: application
- application1: application1
- application2: application2
- application3: application3
- application4: application4
- application5: application5
- webSocketAddress: webSocketAddress
- webSocketPath: webSocketPath

An "Add Param" button is located at the bottom of the list.

⚠ 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

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 Caravasile

Dragos Caravasile

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



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 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
```

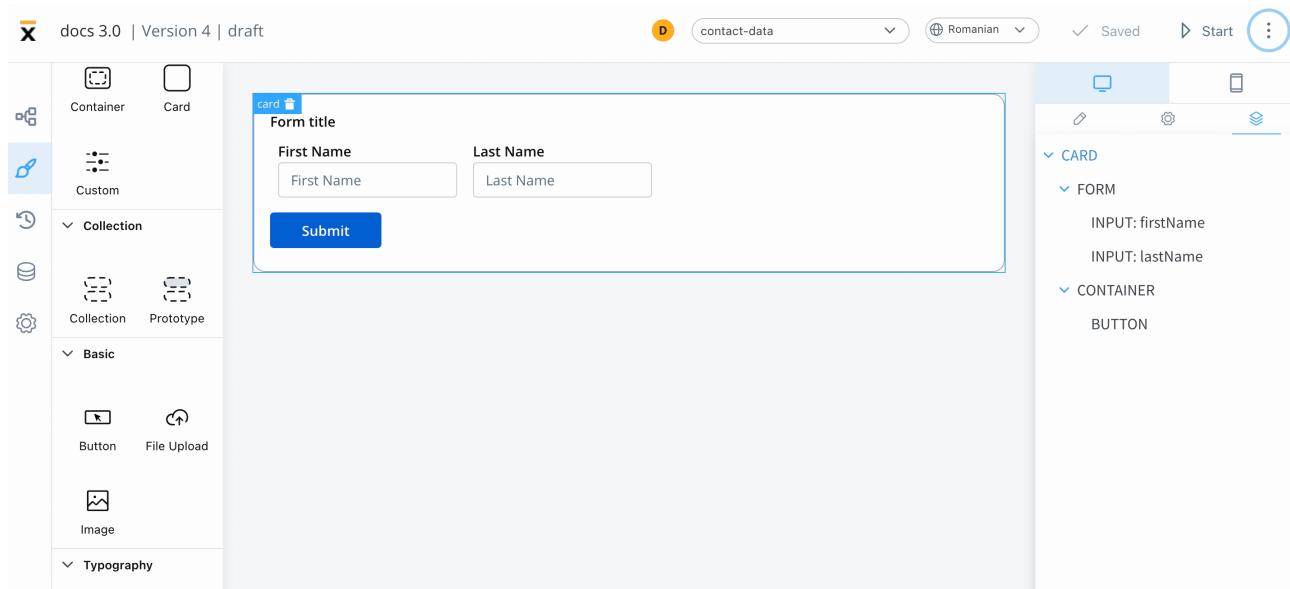
Replace Values

Advanced configuration

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Root components / Container

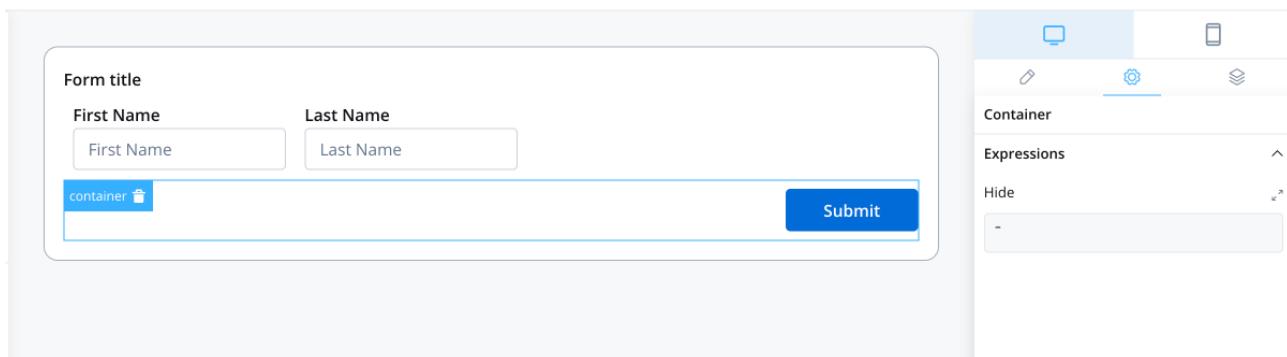
A container is a versatile element that allows you to group components and align them as desired.



The following properties can be configured in the container:

Settings

- **Expressions (Hide)** - JavaScript expressions used to hide components when they evaluate to true



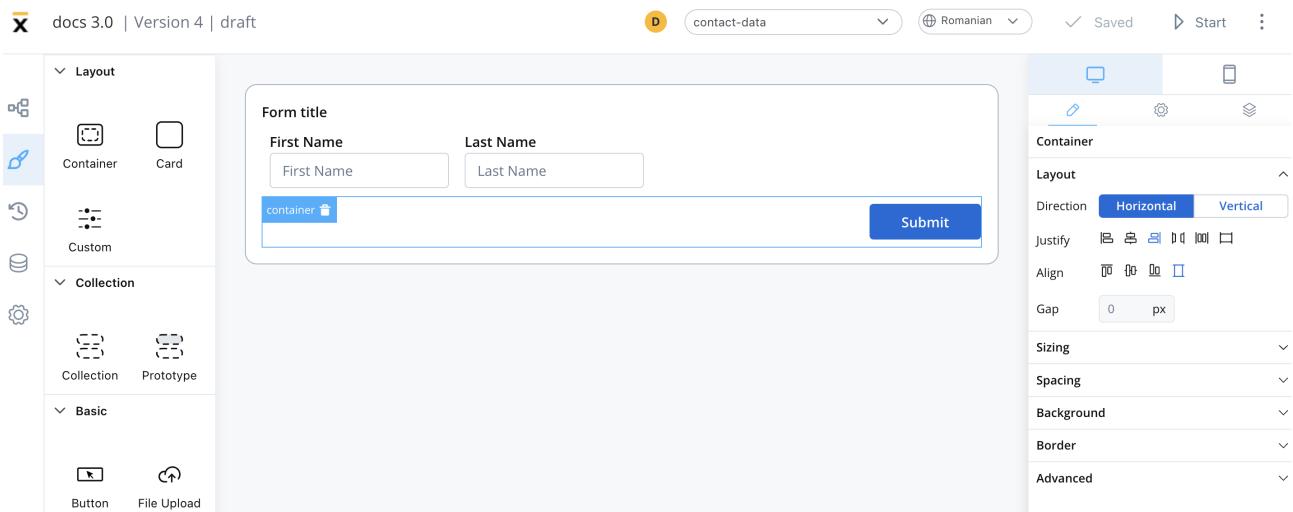
Styling

- **Layout** - this property is available for components that group children and includes the following options:
 - Direction - Horizontal / Vertical (for example, select *Horizontal*)
 - Justify (H) - (for example, select *end*)
 - Align (V) - this option allows you to align components vertically
 - Gap - you can set the gap between components

More layout demos available below:

» [Layout Demos](#)

When you apply the above properties, you can generate the following output, with the button appearing on the right side of the container, underneath the form with three form elements:



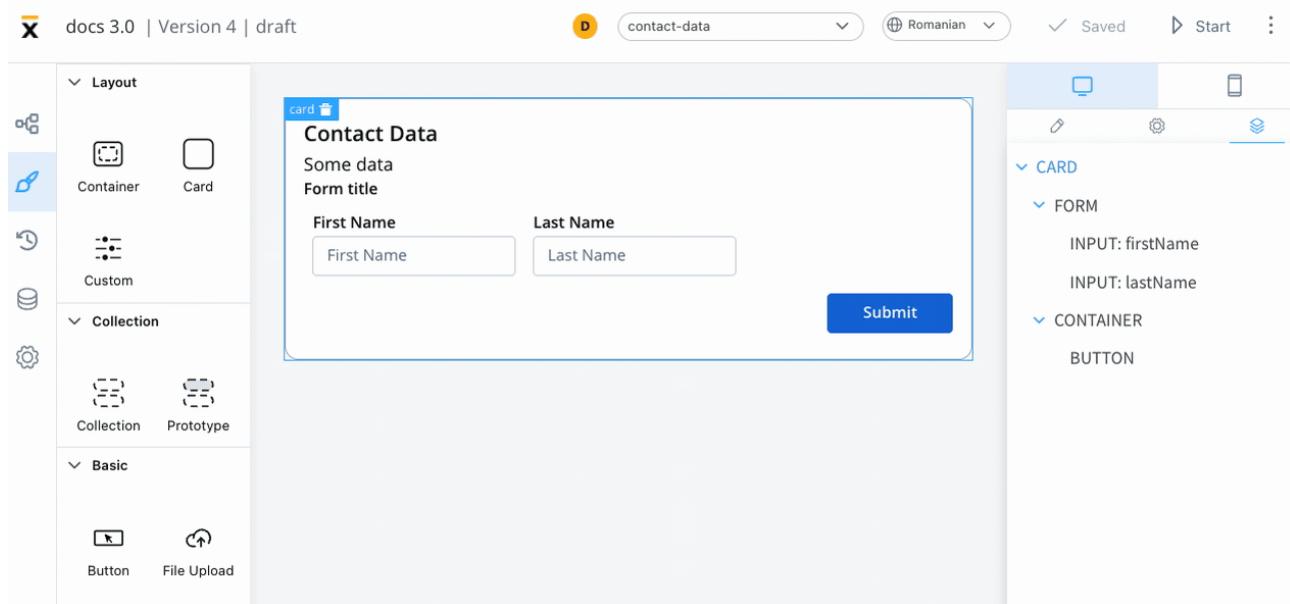
For more information about styling and layout configuration, check the following section:

» [UI Designer](#)

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Root components / Card

A card is a graphical component that allows grouping and alignment of other components. It can also include an accordion element for expanding and collapsing content.



The following properties that can be configured:

Settings

- **Message** - a valid JSON that describes the data pushed to the frontend application when the process reaches a specific user task
- **Title** - the title of the card
- **Subtitle** - the subtitle of the card
- **Card style** - you can choose between a border or raised style
- **Has accordion?** - this feature allows you to add a Bootstrap accordion, which organizes content within collapsible items and displays only one collapsed item at a time

⚠ CAUTION

Accordion element is not available for mobile.

The screenshot shows the FLOWX.AI interface with the 'Card' building block selected. At the top, there are icons for desktop and mobile device preview, a pencil for edit mode, a gear for settings, and a three-dot menu. Below these are sections for 'Card' and 'Message'. The 'Message' section contains a text input field with a placeholder '-' and a double-headed arrow icon. Under the 'Properties' section, there are fields for 'Title' (placeholder 'eg. title') and 'Subtitle' (placeholder 'eg. subtitle'). A 'Card style' section shows two options: 'border' (selected) and 'raised'. At the bottom, there is a checkbox labeled 'Has Accordion'.

Styling

- **Layout** - This property is available for components that group children and includes the following options:
 - Direction - Horizontal / Vertical (for example, select *Vertical*)
 - Justify (H) - (for example, select *center*)
 - Align (V) - this option allows you to align components vertically
 - Gap - you can set the gap between components

More layout demos available below:

» [Layout Demos](#)

This example will generate a card with the following layout configuration:

The screenshot shows the FLOWX.AI interface with the following details:

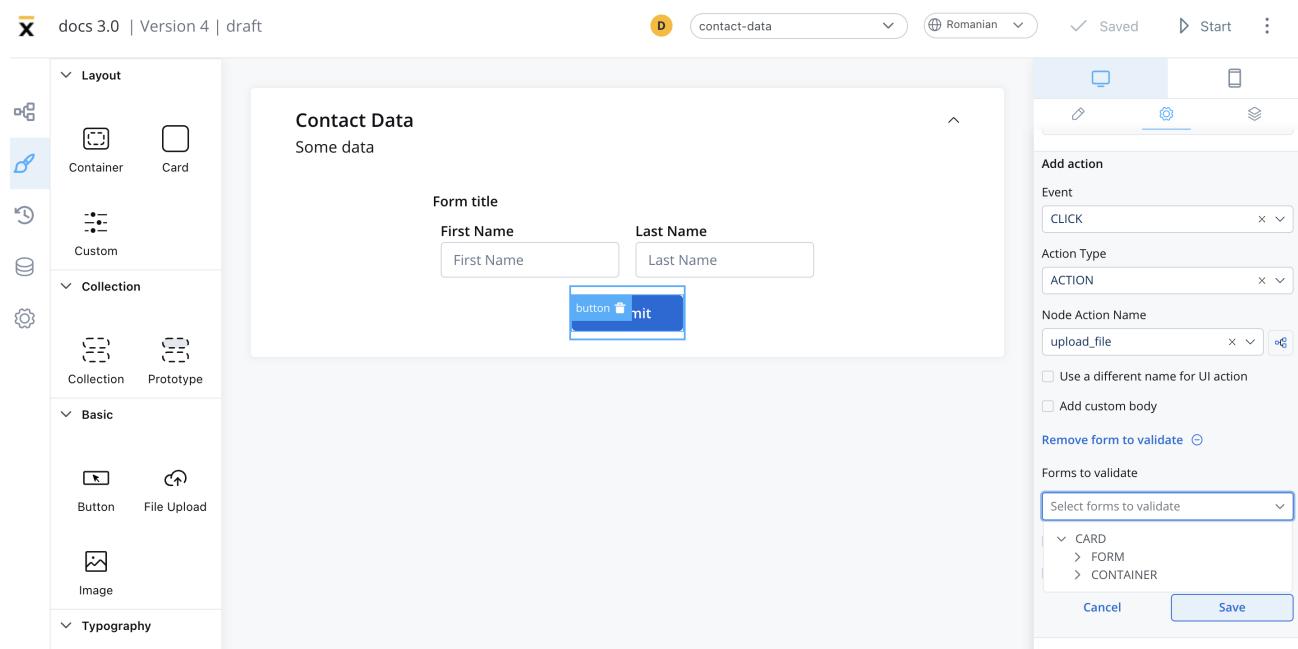
- Header:** docs 3.0 | Version 4 | draft, contact-data, Romanian, Saved, Start.
- Sidebar:** Tools for Layout (Container, Card, Custom), Collection (Collection, Prototype), Basic (Button, File Upload).
- Card Preview:** A card titled "Contact Data" containing "Some data". It has a "Form title" section with "First Name" and "Last Name" fields, and a "Submit" button.
- Right Panel (Layout Configuration):**
 - Card:** Card icon.
 - Layout:** Direction: **Horizontal** (selected), Vertical, Justify: center, Align: center, Gap: 0 px.
 - Sizing:** Fit W: auto.
 - Spacing:** Top: 0, Bottom: 0, Left: 0, Right: 0.

For more information about styling and layout configuration, check the following section:

» [UI Designer](#)

Validating elements

To validate all form elements under a card, you need to set the key of the form/element on the property of the button: *Forms To Validate*.

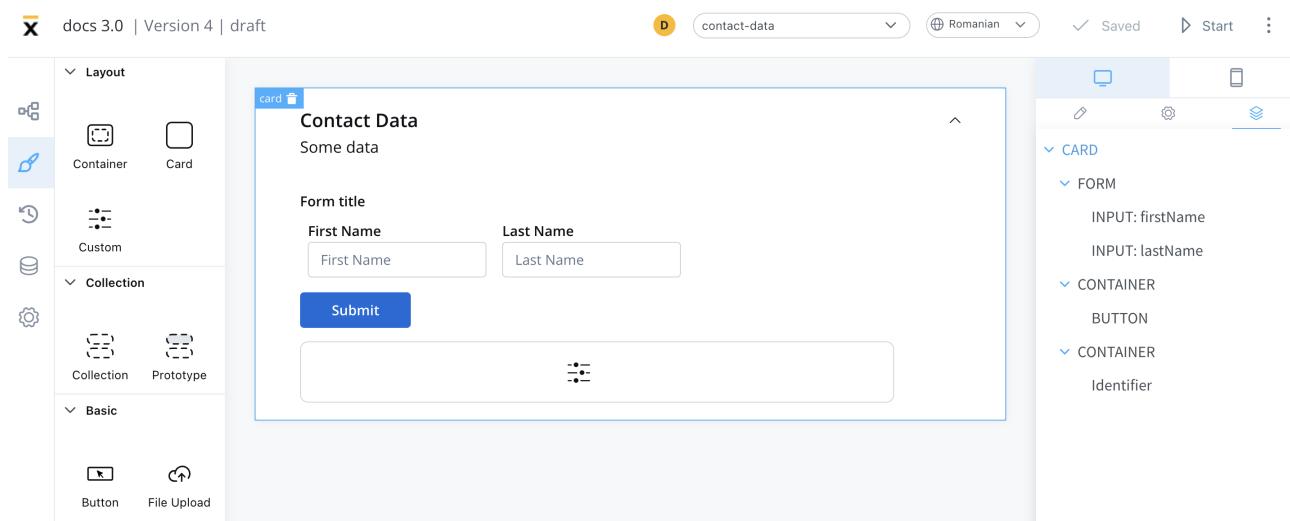


Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Root components / Custom

Custom components are 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.



The properties that can be configured are as follows:

- **Identifier** - this will enable the custom component to be displayed in the component hierarchy and what actions are available for the component
- **Input keys** - used to define the process model paths from which the components will receive their data

- **UI Actions** - actions defined here will be made available to the custom component

The screenshot shows the FLOWX.AI Building Blocks interface. At the top, there are icons for a computer monitor and a smartphone. Below them are icons for a gear (Custom) and a stack of three squares (Input Keys). The main area is divided into sections:

- Custom**: A section labeled "Identifier" contains a text input field with the placeholder "Identifier". To the right of the input field are edit and delete icons.
- Input Keys**: A section containing a button labeled "CustomComponent" with edit and delete icons. Below it is a button labeled "Add an option" with a plus sign icon.
- UI Action**: A section containing a button labeled "Add UI action" with a plus sign icon.

Display of User Interface Elements

When a process instance is initiated, the web application receives all the UI elements that can be displayed in the process under the `templateConfig` key.

When a user task is reached in the process instance, the **events-gateway** receive requests, triggering it to display the associated UI element.

Example:

1. Starting a process:

- The following is an example of starting a process instance via a **POST** request to

```
{{processUrl}}/api/internal/process/DemoProcess/start:
```

```
{
  "processDefinitionName" : "DemoProcess",
  "tokens" : [ {
    "id" : 662631,
    "startNodeId" : null,
    "currentNodeId" : 662807,
    "currentnodeName" : null,
    "state" : "ACTIVE",
    "statusCurrentNode" : "ARRIVED",
    "dateUpdated" : "2023-02-09T12:23:19.464155Z",
    "uuid" : "ae626fda-8166-49e8-823b-fe24f36524a7"
  } ],
  "state" : "CREATED",
  "templateConfig" : [ {
    "id" : 630831,
    "flowxUuid" : "80ea0a85-2b0b-442a-a123-2480c7aa2dce",
    "nodeDefinitionId" : 662856,
    "uiDefinitions" : [
      {
        "id" : 662856,
        "name" : "User Task 1"
      }
    ]
  } ]
}
```

```
"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" : "d985d128-ae45-4408-a643-1dd026a644d3",
    "generalData" : null,
    "backCounter" : 0,
    "startedByActionId" : null,
    "subProcesses" : null,
    "subprocessesUuids" : null,
    "baseUrl" : null
}
```

2. The following is an example of a progress message:

```
{
    "progressUpdateDTO": {
        "processInstanceUuid": "5f24c66f-04a7-433a-b64a-
a765d3b8121a",
        "tokenUuid": "11c32ba6-b3e7-4267-9383-25d69b26492c",
        "currentNodeId": 662856
    }
}
```

3. **ProgressUpdateDto** will trigger the **SDK** to search for the UI element having the same **nodeId** as the one from the web socket progress event
4. Additionally, it will ask for data and actions that are required for this component via a GET request `{{processUrl}}/api/process/5f24c66f-04a7-433a-b64a-a765d3b8121aa/data/662856`

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Collection / Collection Prototype

Description

This additional container type is needed to allow multiple prototypes to be defined for a single **Collection**. This allows elements from the same collection to be displayed differently.

For example, suppose you are creating a piece of UI in which the user is presented a list of possible products from which to choose, but you want one of the products to be highlighted as the recommended one. This example requires a collection with two **collection prototypes** (one for the normal product and one for the recommended one).

Configurable properties:

1. **Prototype Identifier Key** - the key where to look in the iterated object to determine the prototype to be shown - in the below example the key is "type"
2. Prototype Identifier Value - the value that should be present at the **Prototype Identifier Key** when this **COLLECTION_PROTOTYPE** should be displayed - in the below example the value is "normal" or "recommended"

Example

The screenshot shows the FLOWX.AI interface for a project titled "test-collections" (version 1, draft). The left sidebar contains a navigation menu with sections like Layout, Collection, and Basic, each with various icons for different building blocks. The main workspace displays a "Collection prototype example" with two items. The first item is a "Container" with a blue header bar labeled "collection". Inside, there are two cards, each containing the placeholder text "\${name}" and "\${description}". The second item is a "Card" with a yellow border, also containing the same placeholder text. On the right side, there is a toolbar with icons for preview, settings, and export. Below the toolbar, the "COLLECTION" section of the sidebar is expanded, showing the "COLLECTION_PROTOTYPE" section which includes "IMAGE", "TEXT", and another "TEXT" entry. The "IMAGE" entry is highlighted with a blue border.

The screenshot shows the FLOWX.AI interface for creating a collection prototype. On the left, a sidebar lists 'Layout' components: Container, Card, Custom, Collection (selected), and Basic. In the center, a preview area titled 'Collection prototype example' shows two cards with placeholder text: '\${name}' and '\${description}'. The first card has a blue border and is labeled 'collection_prototype'. The second card has a yellow border. On the right, a panel titled 'Prototype Settings' contains 'Flowx props' for 'Prototype Identifier Key' (type: normal) and 'Prototype Identifier Value' (normal). It also includes a section for 'ui Actions' with an 'Add ui action' button.

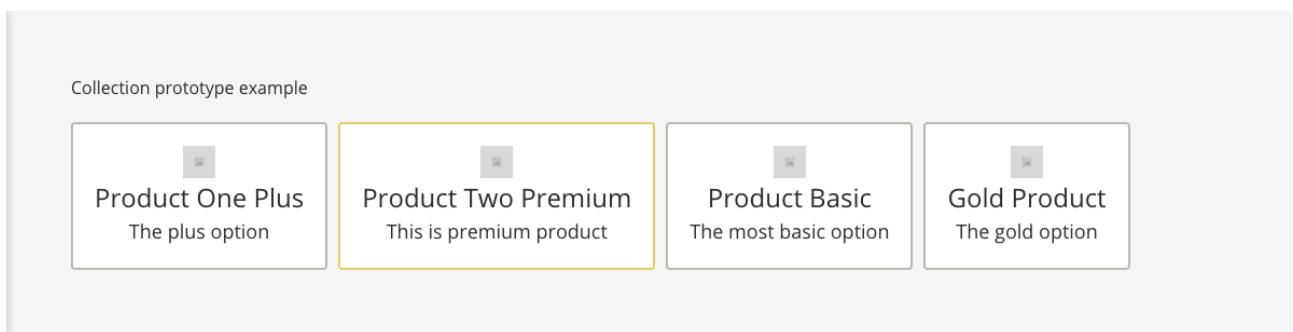
This screenshot is similar to the one above, but the 'Prototype Identifier Value' is set to 'recommended' instead of 'normal'. The rest of the interface remains the same, showing the collection prototype example and the prototype settings panel.

Source collection data example for products:

```
products: [
  {
    name: 'Product One Plus',
    description: 'The plus option'
    type: 'normal'
  },
  {
    name: 'Product Two Premium',
  }
]
```

```
        description: 'This is premium product'
        type: 'recommended',
    },
{
    name: 'Product Basic',
    description: 'The most basic option'
    type: 'normal'
},
{
    name: 'Gold Product',
    description: 'The gold option'
    type: 'normal',
}
]
```

The above configuration will render:



Adding elements with UI Actions

There are a few differences you need to take into consideration when configuring elements that make use of **UI Actions** inside a **Collection Prototype**.

To showcase these differences, we'll use the next example:



We have a **Collection** with two employees and we want to provide the user with the option of selecting one of the employees (eg. to allow for further processing in the next steps of the process).

Step 1 - Defining the Node Action

To select one employee from the list, we first must add an **Action** to the **User Task Node** this UI is attached to:

Node: collection-1 (ID: 431461)

Actions

Action Edit

ID: 431905
Name: save-item
Order: 1

Timer Expression

Save Data

Automatic: Manual:
Mandatory: Optional:
Repeatable:

Autorun Children?

Allow BACK on this action?

Data to send

selectedEmployee

Add Key

Save

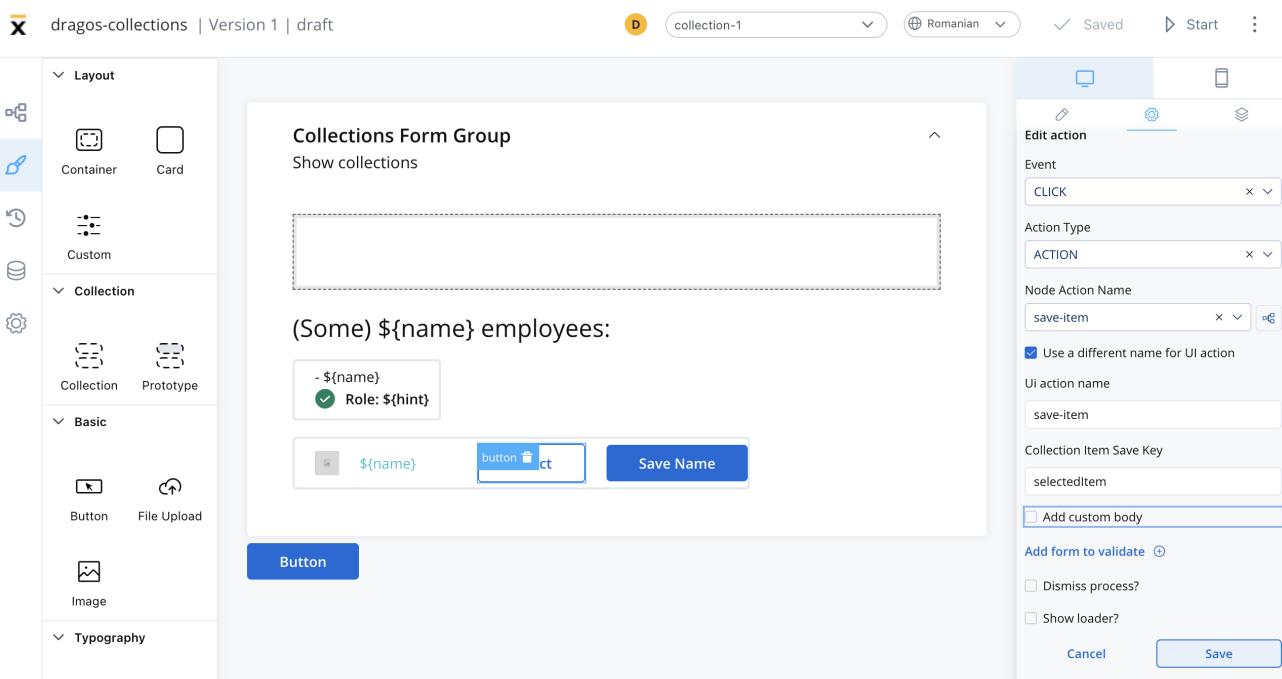
This **save-item** action is **manual** (since it will be triggered by the user) and **optionally** (since selecting an employee is not a requirement to go to the next **Node** in the process).

To allow the user to change his mind about the selected employee, this action is also marked as **Repeatable**.

Keep in mind to check the **Data to send** section. Here we are telling the platform where we want the selected employee (for which the user pressed the **Select** button) to be saved in the **process data**. In this example, we want it to be saved under the `selectedEmployee` key.

Step 2 - Adding the Button & UI Action

Now that we have a **Node Action** defined, we can go ahead and add the **Select** button in the UI of the **User Task** which contains the Employees Collection.



Collection Item Save Key field has an important role in the UI Action configuration of the **Select** button. This field represents how we pass the value of the **Employee** that the user has selected to the **Node Action** that we created in **Step 1**, named `save-item`.

In our example, we set **Collection Item Save Key** to be `selectedEmployee`.

🔥 DANGER

IMPORTANT: `selectedEmployee` key is how we expose the data from the **Collection** to the Node Action. It is **imperative** that the name in the **Collection Item Save Key** is the same as the one used in the **Data to send** input in the Node Action.

The button and UI action are mostly configured as any other Button and UI Action would be configured.

Result

This is how the process data looked before we pressed the **Select** button for an employee:

Process status

Data:

```
processInstanceId: 483001
processData: Object {"companies": [{"employees": [{"name": "Mihai Saru", "imageSrc": "https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Svelte_Logo.svg/1702px-Svelte_Logo.svg.png", "type": "color"}, {"name": "John Doe", "imageSrc": "https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Svelte_Logo.svg/1702px-Svelte_Logo.svg.png", "type": "color"}], "id": 1}, {"id": 1}], tokenStatus: "ACTIVE", tokenType: "PROCESS", tokenId: 483051, tokenUuid: "ae109887-e5aa-46f9-ba18-668ccce33ce8", webSocketPath: "/ws/updates/process", webSocketStatus: "CONNECTED", webSocketAddress: "ws://public.qa.flowxai.dev/01d5b64d-7c61-4d98-a590-d23336f89f95"}
```

Tokens	Token Status	Status Current Node	Date updated
ae109887-e5aa-46f9-ba18-668ccce33ce8	ACTIVE	EXECUTED_PARTIAL	04 May 2022, 12:09 PM

This is how the process data looks after we selected an employee from the list (notice the new field `selectedEmployee`):

Process status

Data:

```
processInstanceId: 483001
processData: Object {"companies":[{"employees":[{"name":"Mihai Saru","imageSrc":"https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Svelte_Logo.svg/1702px-Svelte_Logo.svg.png","type":"colored"}]}]
tokenId: 483051
selectedEmployee:
  name: "Mihai Saru"
  imageSrc: "https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Svelte_Logo.svg/1702px-Svelte_Logo.svg.png"
  type: "colored"
tokenUuid: "ae109887-e5aa-46f9-ba18-668ccce33ce8"
webSocketPath: "/ws/updates/process"
processInstanceUuid: "01d5b64d-7c61-4d98-a590-d23336f89f95"
webSocketAddress: "wss://public.qa.flowxai.dev/01d5b64d-7c61-4d98-a590-d23336f89f95"
```

Tokens	Token Status	Status Current Node	Date updated
Token uuid <code>ae109887-e5aa-46f9-ba18-668ccce33ce8</code>	ACTIVE	EXECUTED_PARTIAL	04 May 2022, 12:09 PM

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Buttons

There are two types of buttons available, each with a different purpose. These types are:

- Basic button
- File upload button

▼ Basic



Button



File Upload

Basic button

Basic buttons are used to perform an action such as unblocking a token to move forward in the process, sending an OTP, and opening a new tab.

Configuring a basic button

When configuring a basic button, you can customize the button's settings by using the following options:

- **Properties**
- **UI action**
- **Button styling**

Sections that can be configured regarding general settings:

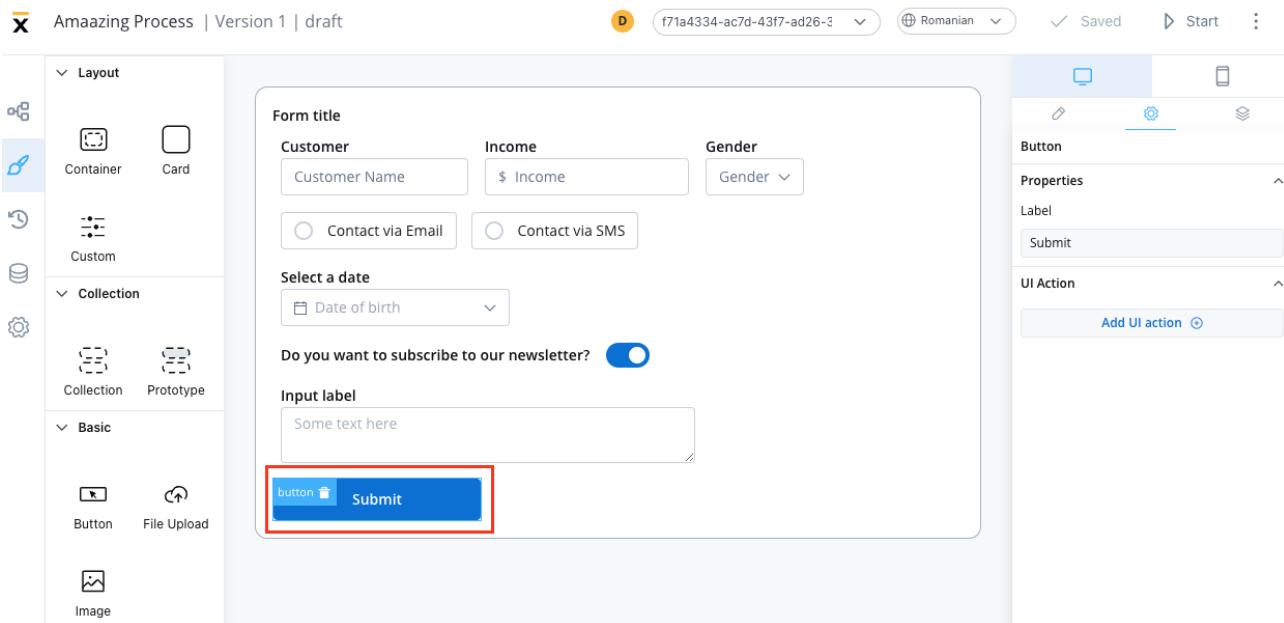
Properties

- **Label** - it allows you to set the label that appears on the button

UI action

Here, you can define the UI action that the button will trigger.

- **Event** - possible value: CLICK
- **Action Type** - select the action type



More details on how to configure UI actions can be found [here](#).

Button styling

Properties

This section enables you to select the type of button using the styling tab in the UI Designer. There are four types available:

- Primary
- Secondary
- Ghost
- Text

The screenshot shows a UI builder interface. On the left, there is a form with various input fields and controls. The form includes sections for 'Form title', 'Customer' (with fields for 'Customer Name'), 'Income' (with field '\$ Income'), 'Gender' (with dropdown), 'Contact via Email' and 'Contact via SMS' (with radio buttons), 'Select a date' (with date picker), and a newsletter subscription toggle. At the bottom is a blue 'Submit' button with a trash icon. On the right, there is a properties panel for a 'Button' element. The properties panel has tabs for 'Button' (selected), 'Properties', 'Sizing', and 'Spacing'. Under 'Properties', the 'Type' is set to 'primary'. Under 'Sizing', 'Fit W' is 'fixed' with a width of '200 px', and 'Fit H' is 'fixed' with a height of '40 px'. Under 'Spacing', the grid shows values: top/bottom 0, left 0, right 0, and gutter 0.

!(INFO)

For more information on valid CSS properties, click [here](#)

Icons

To further enhance the Button UI element with icons, you can include the following properties:

- **Icon Key** - the key associated in the Media library, select the icon from the **Media Library**
- **Icon Color** - select the color of the icon using the color picker

!(INFO)

When setting the color, the entire icon is filled with that color, the SVG's fill. Avoid changing colors for multicolor icons.

- **Icon Position** - define the position of the icon within the button:
 - Left
 - Right
 - Center

!(INFO)

When selecting the center position for an icon, the button will display the icon only.

The form includes the following fields:

- Loan amount: A slider set to 255000 \$.
- Form title: A field labeled "Form title".
- Down payment: A slider set to 38250 \$.
- Form title: A second field labeled "Form title".
- Loan type: A dropdown menu showing "USDA".
- A blue button at the bottom left with a white circular icon.

By utilizing these properties, you can create visually appealing Button UI elements with customizable icons, colors, and positions to effectively communicate their purpose and enhance the user experience.

File upload

This button will be used to select a file and do custom validation on it. Only the Flowx props will be different.

Configuring a file upload button

When configuring a file upload button, you can customize the button's settings by using the following options:

- **Properties**
- **UI action**
- **Button styling**

Sections that can be configured regarding general settings:

Properties

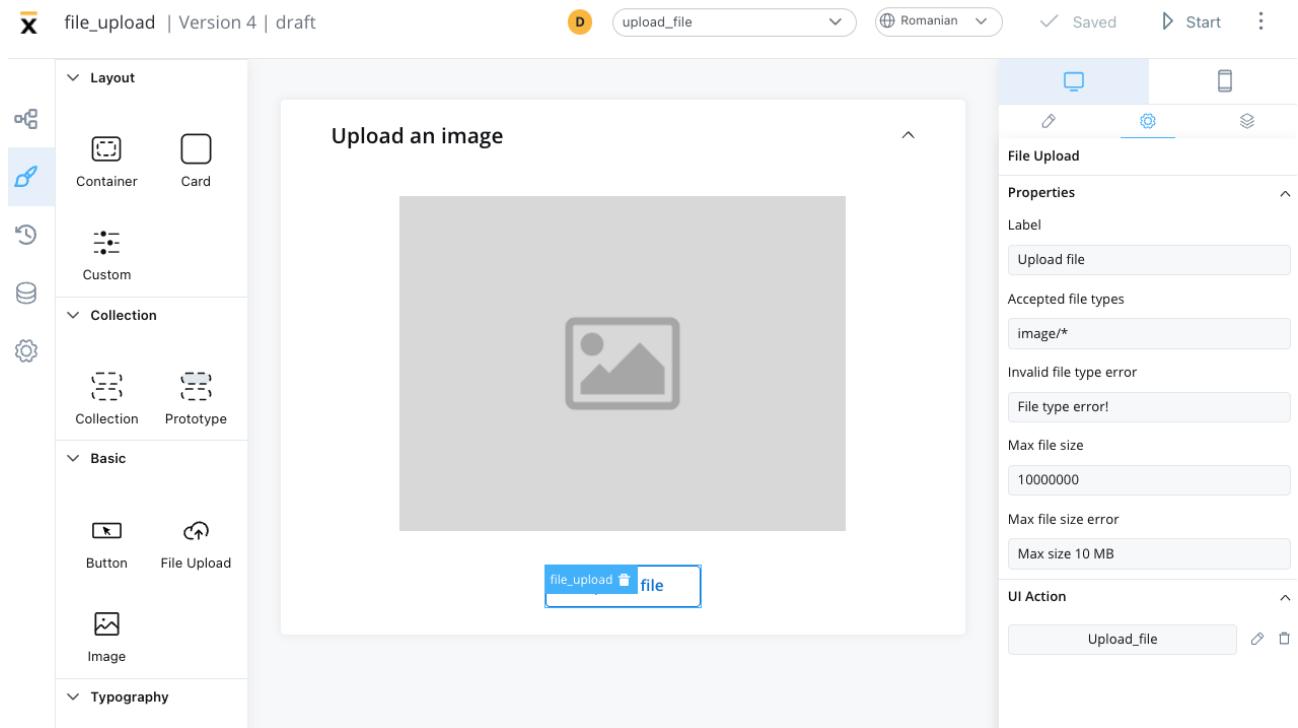
- **Label** - it allows you to set the label that appears on the button
- **Accepted file types** - the accept attribute takes as its value a string containing one or more of these unique file type specifiers, **separated by commas**, may take the following forms:

Value	Definition
audio/*	Indicates that sound files are accepted

Value	Definition
image/*	Indicates that image files are accepted
video/*	Indicates that video files are accepted
MIME type with no params	Indicates that files of the specified type are accepted
string starting with U+002E FULL STOP character (.) (for example, .doc, .docx, .xml)	Indicates that files with the specified file extension are accepted

- **Invalid file type error**
- **Max file size**
- **Max file size error**

Example of an upload file button that accepts image files:



UI action

Here, you can define the UI action that the button will trigger.

- **Event** - possible value: `CLICK`
- **Action Type** - select the action type

The screenshot shows the FLOWX.AI interface for building process flows. On the left, there is a preview window titled "Upload an image" showing a placeholder for an uploaded file. Below the preview is a "file_upload" button with a "file" icon. On the right, the configuration panel for the "File Upload" component is displayed, divided into sections for "Properties" and "UI Action". The "Properties" section includes fields for "Label" (set to "Upload file"), "Accepted file types" (set to "image/*"), "Max file size" (set to "10000000"), and "Max file size error" (set to "Max size 10 MB"). The "UI Action" section, which is highlighted with a red border, contains a "UI Action" dropdown set to "Upload_file", an "Event" dropdown set to "CLICK", and an "Action Type" dropdown set to "ACTION". Other options in the "Action Type" dropdown include "DISMISS", "ACTION", "START_PROCESS_INHERIT", "UPLOAD", and "EXTERNAL". There are also checkboxes for "Add form to validate" and "Dismiss process?", and buttons for "Cancel" and "Save".

INFO

More details on how to configure UI actions can be found [here](#).

Button styling

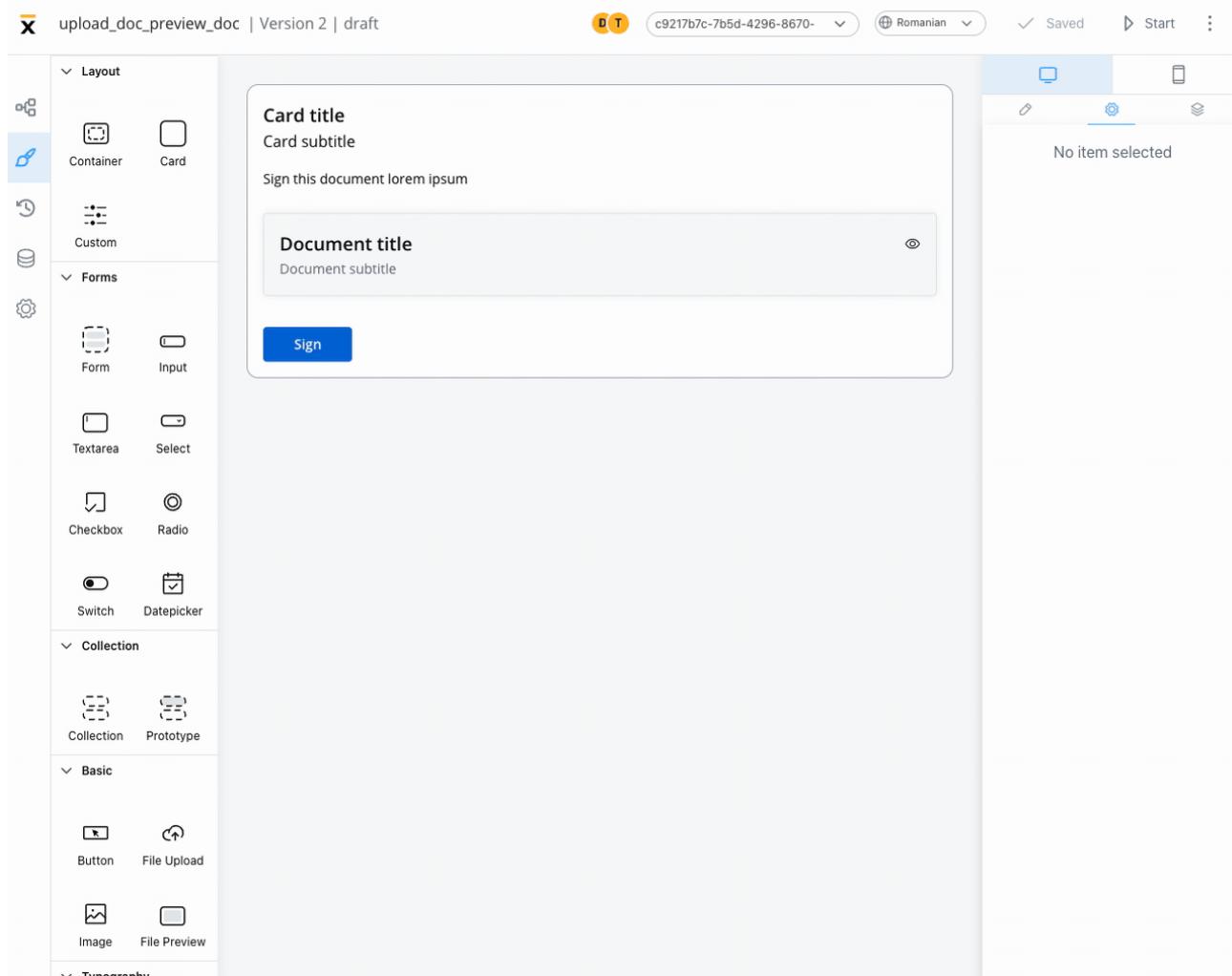
The file upload button can be styled using valid CSS properties (more details [here](#))

[Was this page helpful?](#)

BUILDING BLOCKS / UI Designer / UI component types / File Preview

What is a File Preview UI element?

The File Preview UI element is a user interface component that enables users to preview the contents of files quickly and easily without fully opening them. It can save time and enhance productivity, providing a glimpse of what's inside a file without having to launch it entirely.

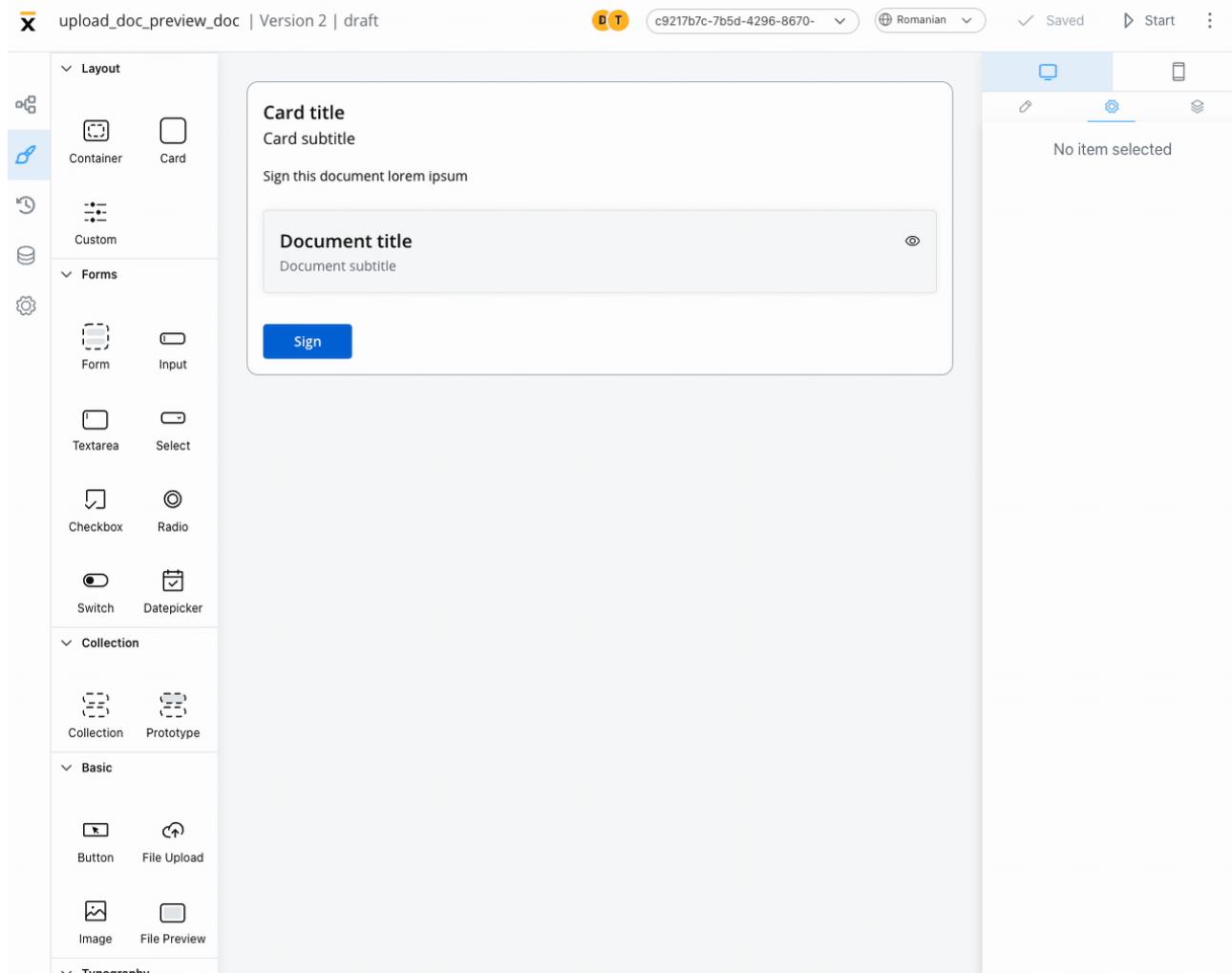


File Preview UI elements offer various benefits such as conveying information, improving the aesthetic appeal of an interface, providing visual cues and feedback or presenting complex data or concepts in a more accessible way.

Configuring a File Preview element

A File Preview element can be configured for both mobile and web applications.

File Preview properties (web)



The File Preview element settings consist of the following properties:

- **Title** - the title of the element (if it is downloaded or shared - the file name should be the title used in preview component)
- **Has subtitle** - the subtitle of the element
- **Display mode** - depending on the selected display method the following properties are available:
 - **Inline → Has accordion:**

- `false` - display preview inline, without expand/collapse option
 - `true` - Default View: Collapsed - display preview inline, with expand/collapse option, by default collapsed
 - `true` - Default View: Expanded - display preview inline, with expand/collapse option, by default expanded
- **Modal** → view icon is enabled
- **Source Type** -
 - **Process Data** - process key where the document is found (creates the binding between the element and process data)
 - **Static** - URL of the document

CAUTION

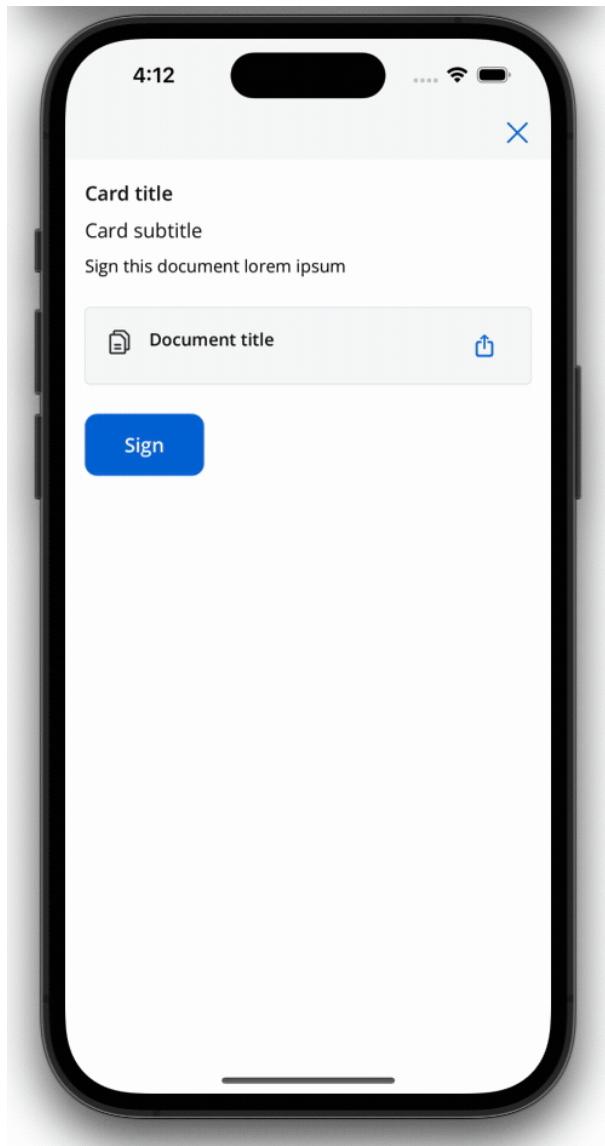
It's worth noting that the inline modal view can raise accessibility issues if the file preview's height exceeds the screen height.

File Preview properties (mobile)

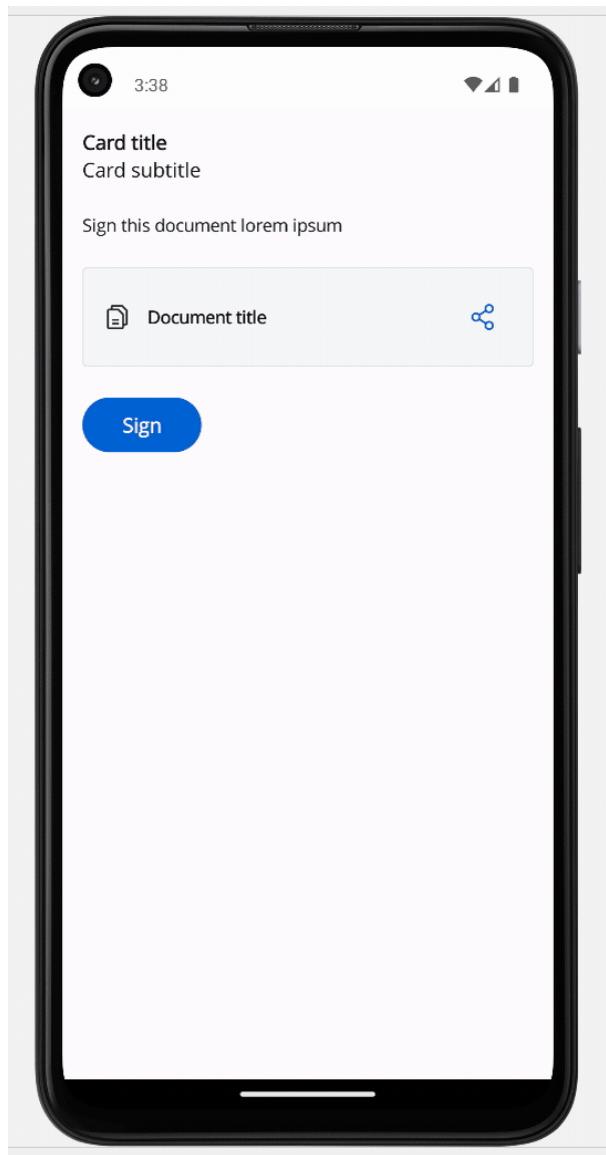
INFO

Both iOS and Android devices support the share button.

iOS



Android



File preview styling

The File Preview styling property enables you to customize the appearance of the element by adding valid CSS properties, for more details, click [here](#).

When drag and drop a File Preview element in UI Designer, it comes with the following default styling properties:

Sizing

- **Fit W - auto**
- **Fit H - fixed / Height - 400 px**

The screenshot shows the FLOWX.AI interface with the following details:

- Left Sidebar (Toolbox):**
 - Layout:** Container, Card, Custom.
 - Forms:** Form, Input, Textarea, Select, Checkbox, Radio, Switch, Datepicker.
 - Collection:** Collection, Prototype.
 - Basic:** Button, File Upload.
- Middle Panel (Preview Area):**

Card title
Card subtitle
Sign this document lorem ipsum

document_preview Document title
Document subtitle

Test PDF

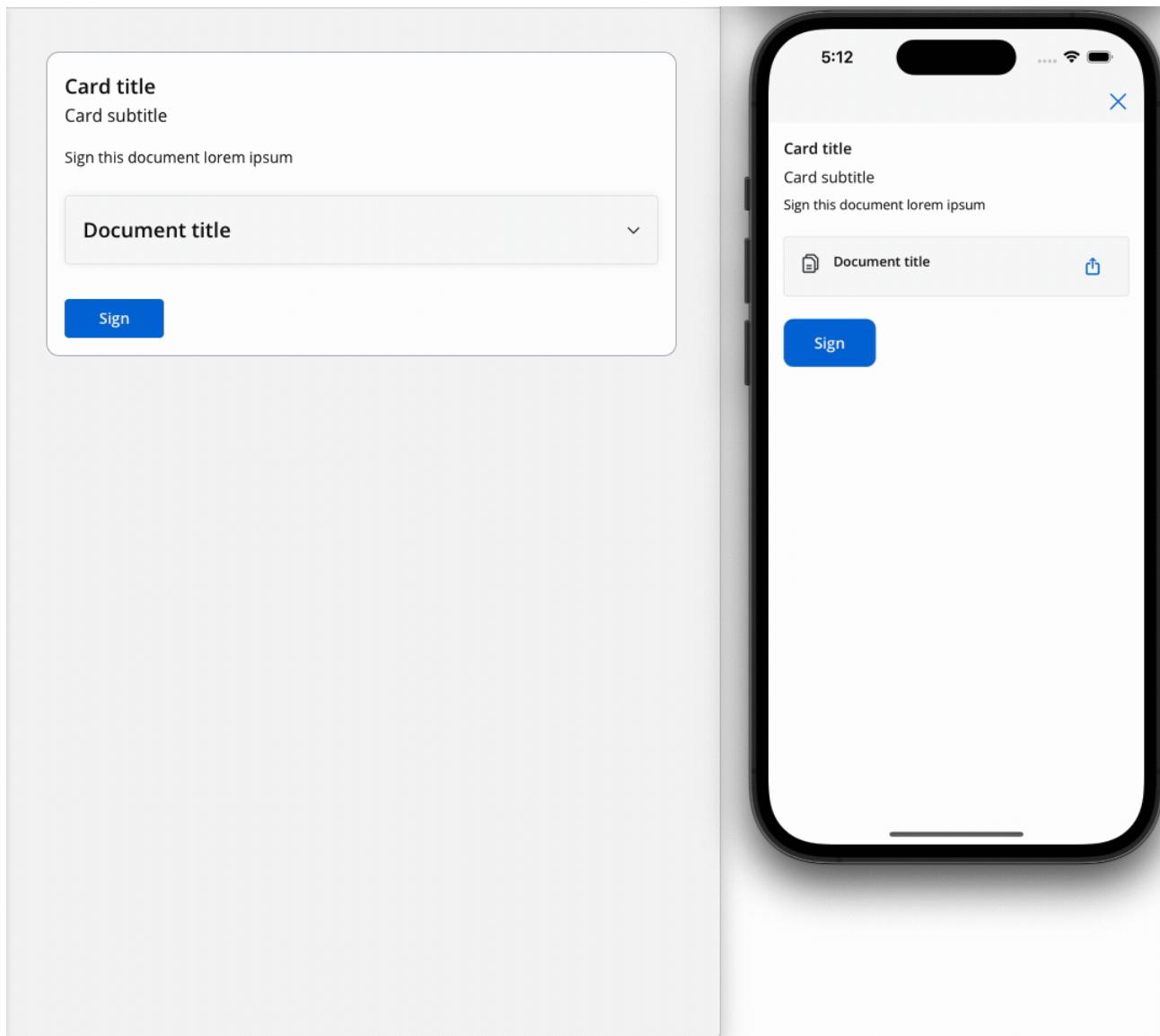
Content area containing Lorem Ipsum text.

Sign button at the bottom.
- Right Panel (Properties):**
 - File Preview** section:
 - Sizing:** Fit W: auto, Fit H: fixed, Height: 400 px.
 - Spacing:** All values set to 0.
 - Typography:** Title color: #1E1E1C, Subtitle color: #1E1E1C.
 - Background:** Color: #1E1E1C.
 - Border:** Radius: 0 px, Width: 0 px, Color: #1E1E1C.
 - Advanced:** Add class: Comma separated class names.

File Preview example

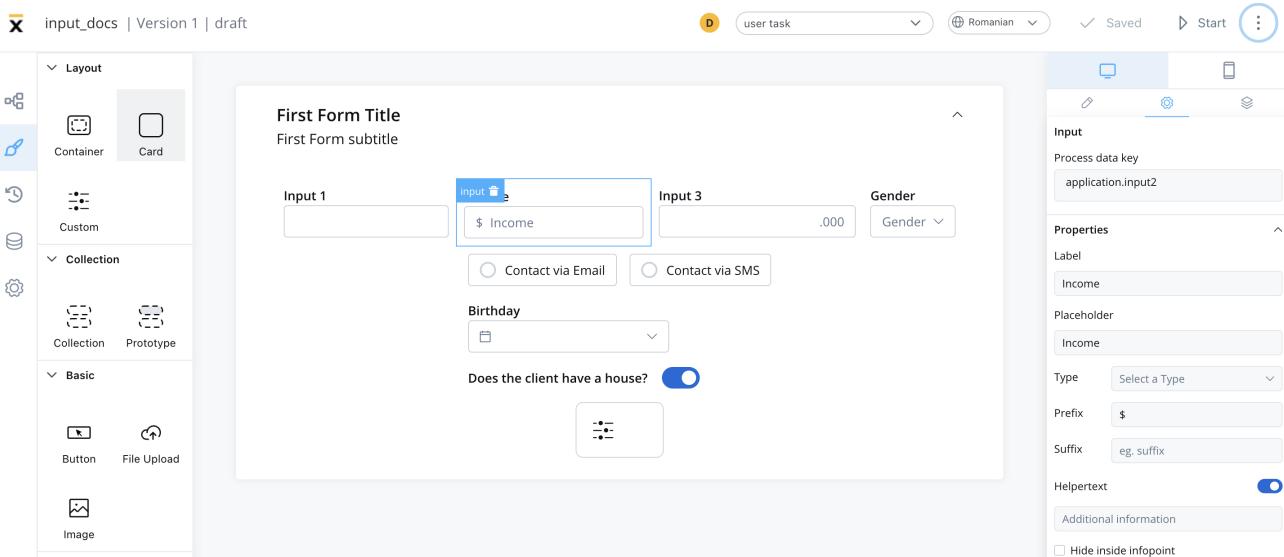
Below is an example of a File Preview UI element with the following properties:

- **Display mode - Inline**
- **Has accordion - True**
- **Default view - Expanded**
- **Source Type - Static**



Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Input



An input field is a form element that enables users to input data with validations and can be hidden or disabled.

Configuring the input element

Input settings

The Input Field offers the following configuration options:

- General
- Properties
- Datasource

- **Validators**
- **Expressions**
- **UI actions**
- **Input styling**

General

- **Process data key** - creates the binding between form element and process data, so it can be later used in **decisions**, **business rules** or **integrations**

Properties

- **Label** - the label that appears on the input field
- **Placeholder** - the placeholder text that appears in the input field when it is empty
- **Type** - the type of data that the input field can accept, such as text, number, email, or password
- **Prefix** - a label that appears as a prefix to the input field
- **Suffix** - a label that appears as a suffix to the input field
- **Helpertext** - additional information about the input field (can be hidden inside an infopoint)

Datasource

The default value for the element can be configured here, this will autofill the input field when you will run the process.

First Form Title ^

First Form subtitle

Income Mortgage Gender

\$ 555 .000 Gender ▾

This is a helper text

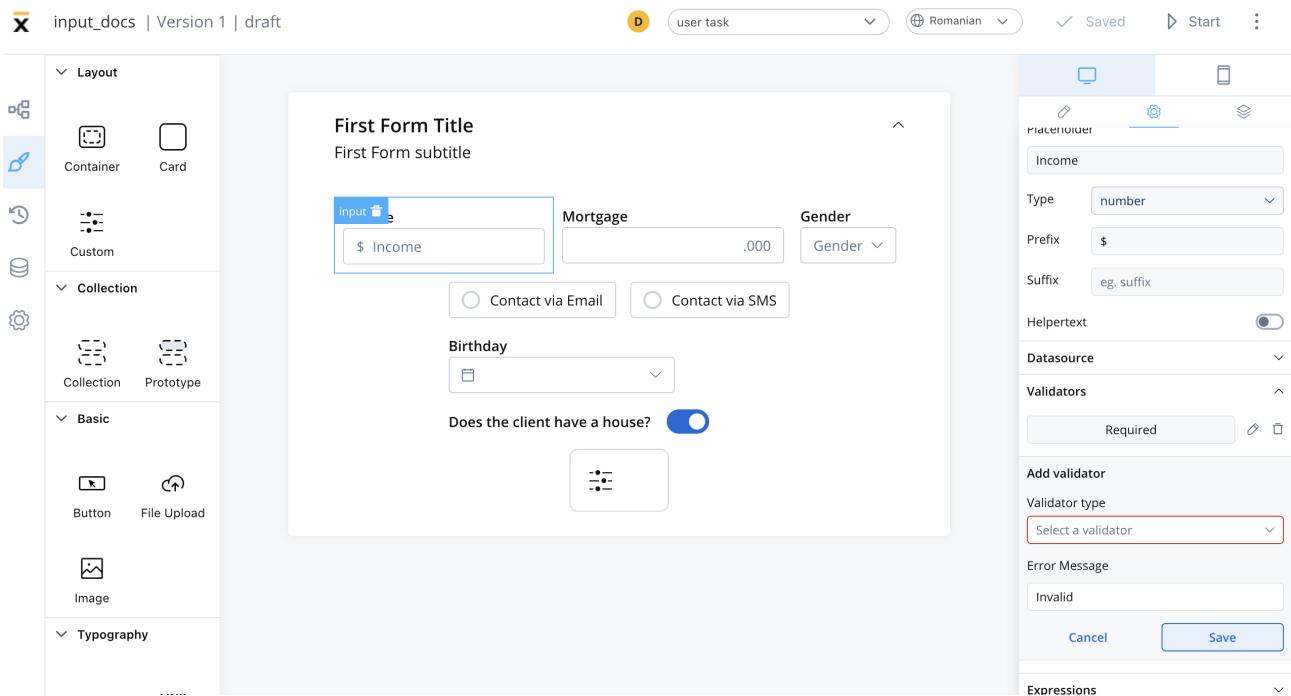
Contact via Email Contact via SMS

Birthday ▼

Does the client have a house?

Validators

There are multiple validators can be added to an input (more details [here](#)).



Expressions

The input field's behavior can be defined using JavaScript expressions for hiding or disabling the element. The following properties can be configured for expressions:

- **Hide** - JavaScript expression used to hide the Input Field when it returns a truthy value
- **Disabled** - JavaScript expression used to disable the Input Field when it returns a truthy value

! INFO

It's important to make sure that disabled fields have the same expression configured under the path expressions → hide.

The screenshot shows the FLOWX.AI UI builder interface. On the left, there's a sidebar with categories like Layout, Collection, and Basic, each containing icons for Container, Card, Custom, Collection, Prototype, Button, File Upload, and Image. The main area displays a form titled "First Form Title" with a subtitle "First Form subtitle". The form contains several input fields: a text input for "Income" with a placeholder "\$ Income", a numeric input for "Mortgage" with a placeholder ".000", a dropdown for "Gender", and two radio buttons for "Contact via Email" and "Contact via SMS". Below these are a date input for "Birthday" and a toggle switch for "Does the client have a house?". To the right of the form is a detailed configuration panel for the "Income" field, which includes sections for Placeholder, Type (number), Prefix (\$), Suffix (eg. suffix), Helpertext (disabled), Datasource (disabled), Validators (disabled), and Expressions. The Expressions section contains the code \${application.key}=='TEST'. The top of the screen shows navigation and status bars: "input_docs | Version 1 | draft", "user task", "Romanian", "Saved", "Start", and a three-dot menu.

UI actions

UI actions can be added to the Input Field to define its behavior and interactions.

- **Event** - possible value: `CHANGE`
- **Action Type** - select the action type

The screenshot illustrates the FLOWX.AI platform's form builder. The left sidebar provides a categorized view of available building blocks. The main workspace displays a form titled "First Form Title" with a subtitle "First Form subtitle". This form includes several input fields: "Income" (with a dollar sign icon), "Mortgage" (with a house icon), "Gender" (with a gender icon), and options for "Contact via Email" or "SMS". A "Birthday" date picker is also present. A toggle switch asks if the client has a house. To the right, a detailed configuration panel is open for the "Income" field, allowing users to set its type as a number, add a prefix (\$), and a suffix (e.g., suffix), and define various validation and hide conditions.

(!) INFO

For more details on how to configure a UI action, click [here](#).

Input styling

Icons

- **Icon Key** - the key associated in the Media library, select the icon from the **Media Library**
- **Icon Color** - select the color of the icon using the color picker

(!) INFO

When setting the color, the entire icon is filled with that color, the SVG's fill. Avoid changing colors for multicolor icons.

You have the option to enhance the Input element by incorporating two types of icons:

- **Left Icon:** You can include an icon on the left side of the Input element. This icon can serve as a visual cue or symbol associated with the input field's purpose or content.
- **Right Icon:** Same as left icon.

By utilizing these two types of icons, you can provide users with a more intuitive and visually appealing experience when interacting with the Input element.

The screenshot shows a 'Enter Personal Information' form with various input fields and a sidebar for styling.

Form Fields:

- First Name: Input field with a person icon and placeholder 'First Name'.
- Last Name: Input field with a person icon and placeholder 'Last Name'.
- Date of birth: Input field with a calendar icon and placeholder 'Placeholder'.
- Employment type: Radio button group with options 'Employed' and 'Pensioner'.
- Save personal information: A toggle switch.
- Loan amount: A slider with values '10000 \$' and '500000 \$'. The current value is '255000 \$'.
- Form title: A text input field.

Properties Sidebar:

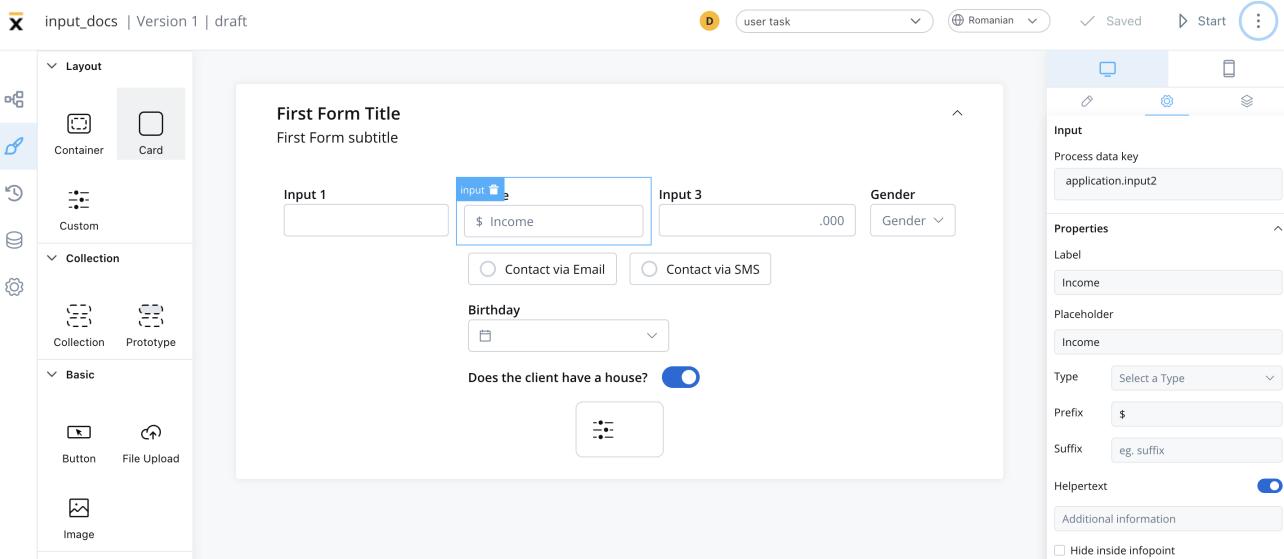
- Input:** Shows mobile device icons.
- Properties:**
 - Left Icon:** Enabled (blue switch). Options: 'Icon key' (Change), 'Icon color' (Set color).
 - Right Icon:** Enabled (blue switch). Options: 'Icon key' (Change), 'Icon color' (Set color).
- Sizing:** Fit W: 'fill'.
- Spacing:** Grid settings: 0, 0, 0, 0.
- Typography:** Set color.

- The Input Field can be styled using valid CSS properties (more details [here](#))

The screenshot shows the FLOWX.AI UI Designer interface. On the left, there's a sidebar with categories: Layout (Container, Card, Custom), Collection (Collection, Prototype), and Basic (Button, File Upload, Image). The main area displays a form titled "First Form Title" with a subtitle "First Form subtitle". The form contains several input fields: a text input for "Income" with a placeholder "\$ Income", a text input for "Mortgage" with a placeholder ".000", a dropdown for "Gender", two radio buttons for "Contact via Email" and "Contact via SMS", a date input for "Birthday", and a toggle switch for "Does the client have a house?". To the right of the form is a panel with tabs for desktop and mobile viewports. The "Input" tab is selected, showing properties for the "Income" field: Process data key "application.input2", Label "Income", Placeholder "Income", Type "number", Prefix "\$", Suffix "eg. suffix", Helpertext (checkbox checked), Datasource (dropdown), and Validators (dropdown). There are also tabs for Properties, Label, Placeholder, Type, Prefix, Suffix, Helpertext, Datasource, and Validators.

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Text area



A text area is a form element used to capture multi-line input from users in a conversational interface. The text area component is typically used for longer inputs such as descriptions, comments, or feedback, providing users with more space to type their responses.

It is an important tool for creating intuitive and effective conversational interfaces that can collect and process large amounts of user input.

Configuring the text area element

Text area settings

The text area offers the following configuration options:

- **General**
- **Properties**
- **Datasource**
- **Validators**

- **Expressions**
- **UI actions**
- **Text area styling**

General

- **Process data key** - creates the binding between form element and process data, so it can be later used in [decisions](#), [business rules](#) or [integrations](#)

Properties

- **Label** - the label of the text area
- **Placeholder** - the placeholder text that appears in the text area
- **Helpertext** - additional information about the text area field (can be hidden inside an infopoint)

Datasource

The default value for the element can be configured here, this will autofill the text field when you will run the process.

Validators

There are multiple validators can be added to a text area element (more details [here](#)).

The screenshot shows the FLOWX.AI builder interface. On the left, there's a sidebar with categories like TEXT, LINK, FORMS, INDICATORS, and a MESSAGE icon. Under FORMS, there are icons for Form, Input, Textarea, Select, Checkbox, Radio, Switch, and Datepicker. The main workspace displays a form titled 'Form title'. It includes fields for 'Customer' (Customer Name, Income, Gender), contact methods (Contact via Email, Contact via SMS), a date selector (Date of birth), a newsletter subscription toggle, and a large text area labeled 'textKey' with placeholder text 'Some text here'. A 'Submit' button is at the bottom. To the right of the form, there's a panel for the 'Textarea' component. It shows configuration options: 'Process data key' set to 'textKey', 'Properties' (Label: 'Input label', Placeholder: 'Some text here', Helpertext: checked), 'Datasource' (Default value: 'eg. Name'), and 'Validators' (with a 'Add a validator' button). The top of the screen has a header with a document icon, the path 'docs_form | Version 1 | draft', a language switch to 'Romanian', a save status, and navigation buttons.

Expressions

The text area's behavior can be defined using JavaScript expressions for hiding or disabling the element. The following properties can be configured for expressions:

- **Hide** - JavaScript expression used to hide the text area when it returns a truthy value
- **Disabled** - JavaScript expression used to disable the text area when it returns a truthy value

INFO

It's important to make sure that disabled fields have the same expression configured under the path expressions → hide.

The screenshot shows the FLOWX.AI UI builder interface. On the left, there's a sidebar with categories like Layout, Collection, and Basic, each containing icons for Container, Card, Custom, Collection, Prototype, Button, File Upload, and Image. The main area displays a form titled "First Form Title" with a subtitle "First Form subtitle". The form includes fields for "Income" (a text input with a placeholder "\$ Income"), "Mortgage" (a text input with a placeholder ".000"), "Gender" (a dropdown menu), "Contact via Email" and "Contact via SMS" (radio buttons), "Birthday" (a date picker), and a toggle switch labeled "Does the client have a house?". To the right of the form is a detailed configuration panel for the "Income" field. It shows settings for Placeholder ("Income"), Type ("number"), Prefix ("\$"), Suffix ("eg. suffix"), Helpertext (a toggle switch), Datasource (a dropdown menu), and Validators (a dropdown menu). The "Validators" section is expanded, revealing an "Expressions" block with the code "\${application.key}=='TEST'". There are also sections for "Hide" and "Disabled". The top of the screen has a header with a project name "input_docs | Version 1 | draft", a status bar indicating "user task", language "Romanian", and a "Saved" status, along with a "Start" button and a more options menu.

UI actions

UI actions can be added to the text area field to define its behavior and interactions.

- **Event** - possible value: `CHANGE`
- **Action Type** - select the action type

The screenshot shows the FLOWX.AI UI Designer interface. On the left, there's a sidebar with categories like TEXT, LINK, Forms, Indicators, and a message icon. The main area displays a form titled "Form title". The form contains fields for "Customer Name", "Income", and "Gender". It also includes two radio buttons for "Contact via Email" and "Contact via SMS", a dropdown for "Select a date" (Date of birth), and a toggle switch for "Do you want to subscribe to our newsletter?". Below these is a large text area with placeholder text "Some text here" and a "Submit" button. To the right of the form, there are several configuration panels: "Helpertext" (with a toggle switch), "Datasource" (with a dropdown for "Default value" set to "eg. Name"), "Validators" (with a "Add a validator" button), "Expressions" (with a "Hide" section containing a condition "\$(textKey) !== 'TEST'", and a "Disabled" section with a dropdown set to "-"), and "UI Action" (with a "Add UI action" button). The top right of the interface shows "f71a4334-ac7d-43f7-ad26-3" (ID), "Romanian" (language), "Saved" (status), "Start" (button), and a three-dot menu.

INFO

For more details on how to configure a UI action, click [here](#).

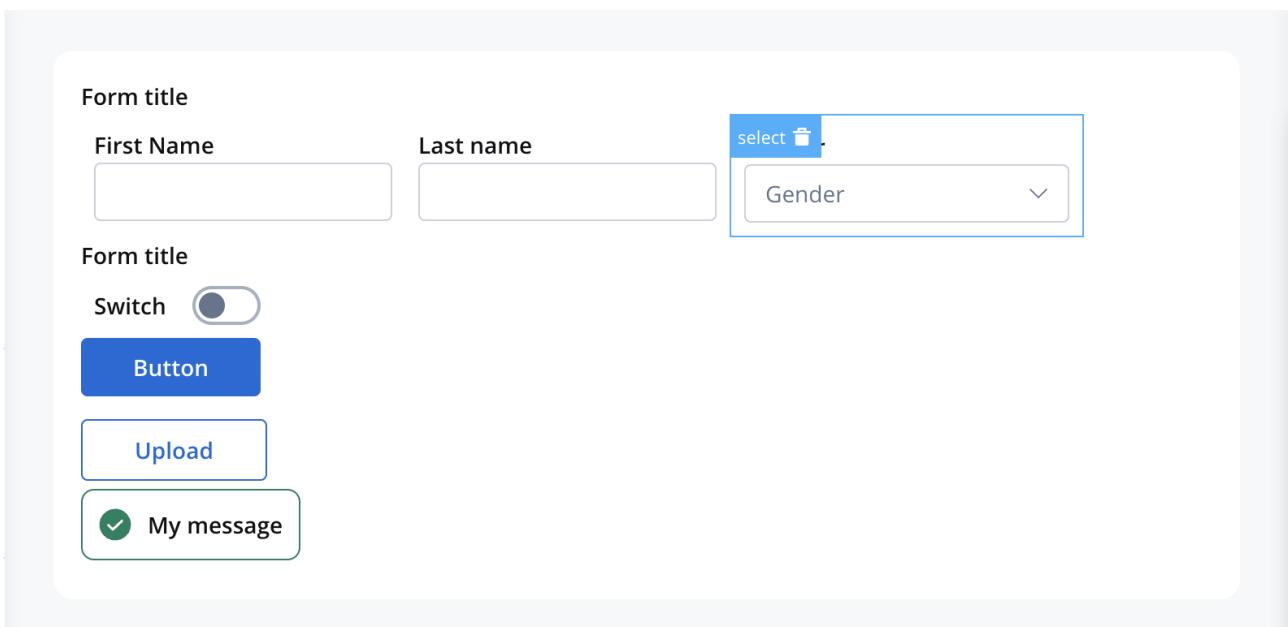
Text area styling

The ability to style the text area element using CSS properties is relevant because it allows you to customize the appearance of the text area to match the overall design of the website or application.

» [UI Designer styling](#)

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Select



The Select form field is an element that enables users to make a choice from a list of predefined options. It consists of multiple values, each of which is defined by a label that is displayed in the dropdown menu, and a code that is saved.

(!) INFO

For instance, you could have a label of "Female" with the value "F" and "Male" with the value "M". This means that when a user selects "Female" in the process instance, the value "F" will be stored for the "Select" key.

Configuring the Select element

Select Settings

These allow you to customize the settings for the Select Field:

- **General**
- **Properties**
- **Datasource**
- **Validators**
- **Expressions**
- **UI actions**
- **Select styling**

General

- **Process data key** - creates the binding between form element and process data so it can be later used in **decisions**, **business rules** or **integrations**

Properties

- **Label** - the label of the select
- **Placeholder** - placeholder when the field has no value
- **Empty message** - text displayed for custom type when no results are found
- **Search for options** - displays a search to filter options
- **Helpertext** - additional information about the select field (can be hidden inside an infopoint)

Datasource

- **Default value** - autofill the select with this value. Going back to the example with Woman label with F value and Man with M to have a default value of Woman we need to configure here F
- **Source Type** - it can be Static, Enumeration, or Process Data
- **Add option** - label - value pairs can be defined here

Validators

There are multiple validators can be added to a select (more details [here](#)).

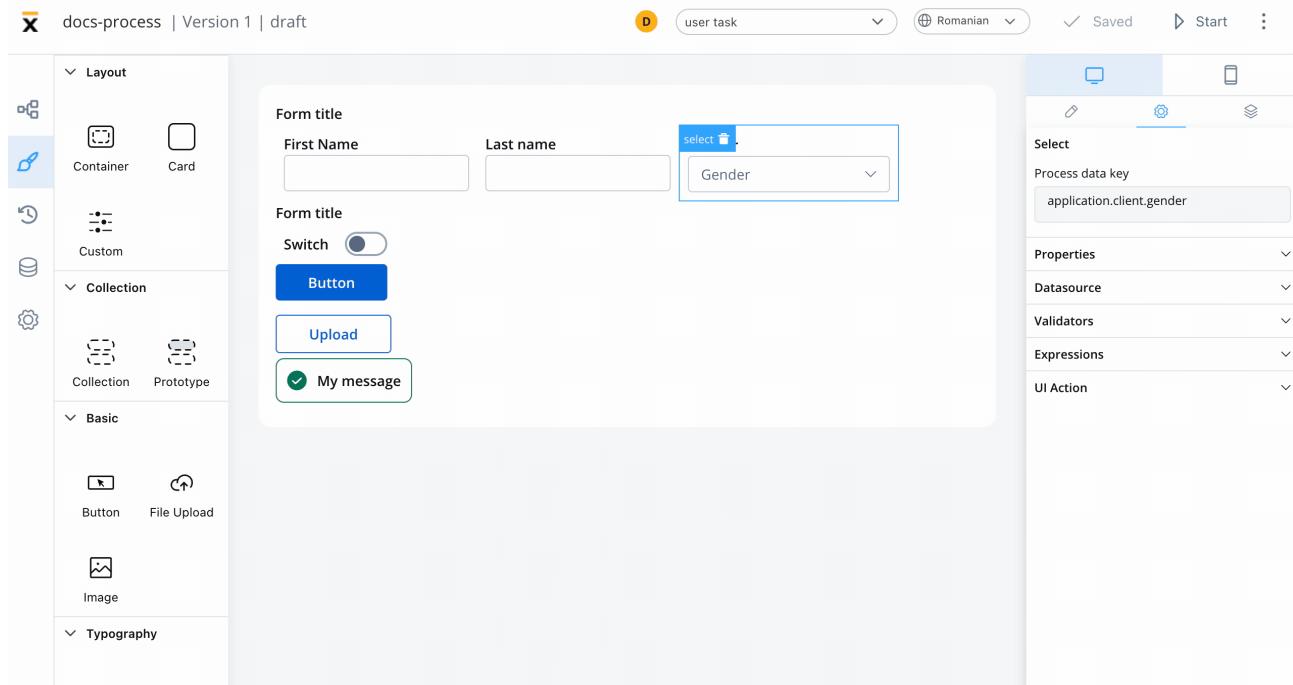
Expressions

The select field's behavior can be defined using JavaScript expressions for hiding or disabling the element. The following properties can be configured for expressions:

- **Hide** - JavaScript expression used to hide the Select Field when it returns a truthy value
- **Disabled** - JavaScript expression used to disable the Select Field when it returns a truthy value

(!) INFO

It's important to make sure that disabled fields have the same expression configured under the path expressions → hide.



UI actions

UI actions can be added to the select element to define its behavior and interactions.

- **Event** - possible value: `CHANGE`
- **Action Type** - select the action type

!(INFO)

For more details on how to configure a UI action, click [here](#).

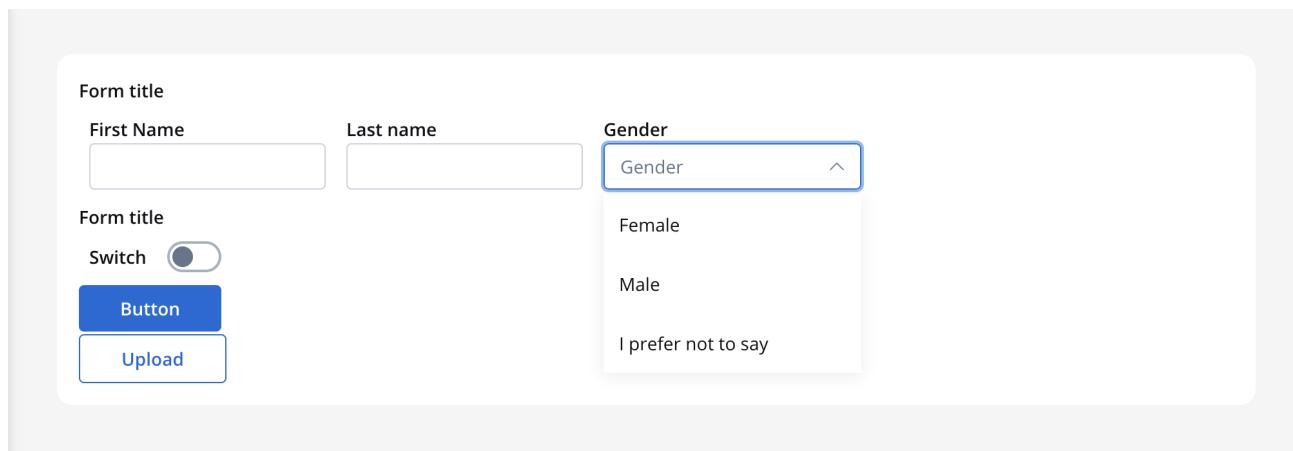
Select styling

Styling the Select field using CSS properties allows you to customize the appearance of the dropdown list and make it more visually appealing and

consistent with the overall design of the website or application.

» UI Designer styling

For example, a FORM element with a **layout** configuration including direction of Horizontal and some inputs, and a select element will look like this:



Icons

When customizing the appearance of a Select UI element that includes an icon, you can utilize the following properties:

- **Icon Key** - the key associated in the Media library, select the icon from the **Media Library**
- **Icon Color** - select the color of the icon using the color picker

(!) INFO

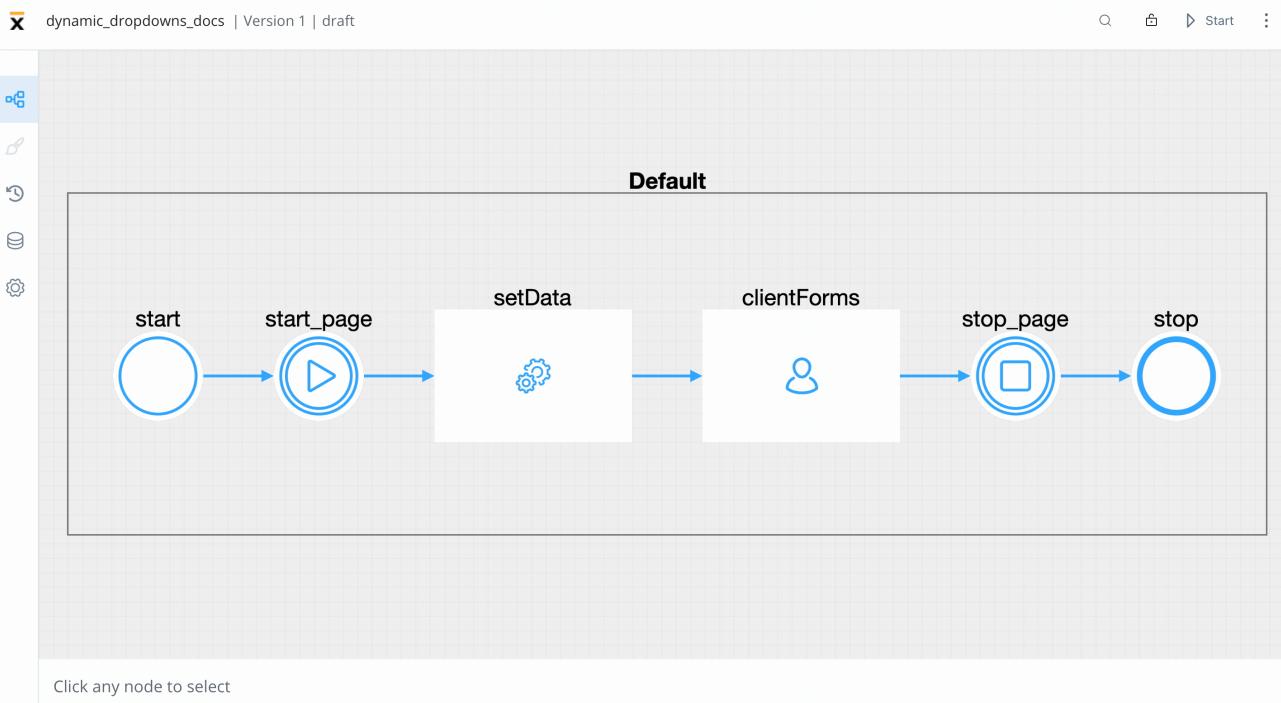
When setting the color, the entire icon is filled with that color, the SVG's fill. Avoid changing colors for multicolor icons.

Example - Dynamic dropdowns

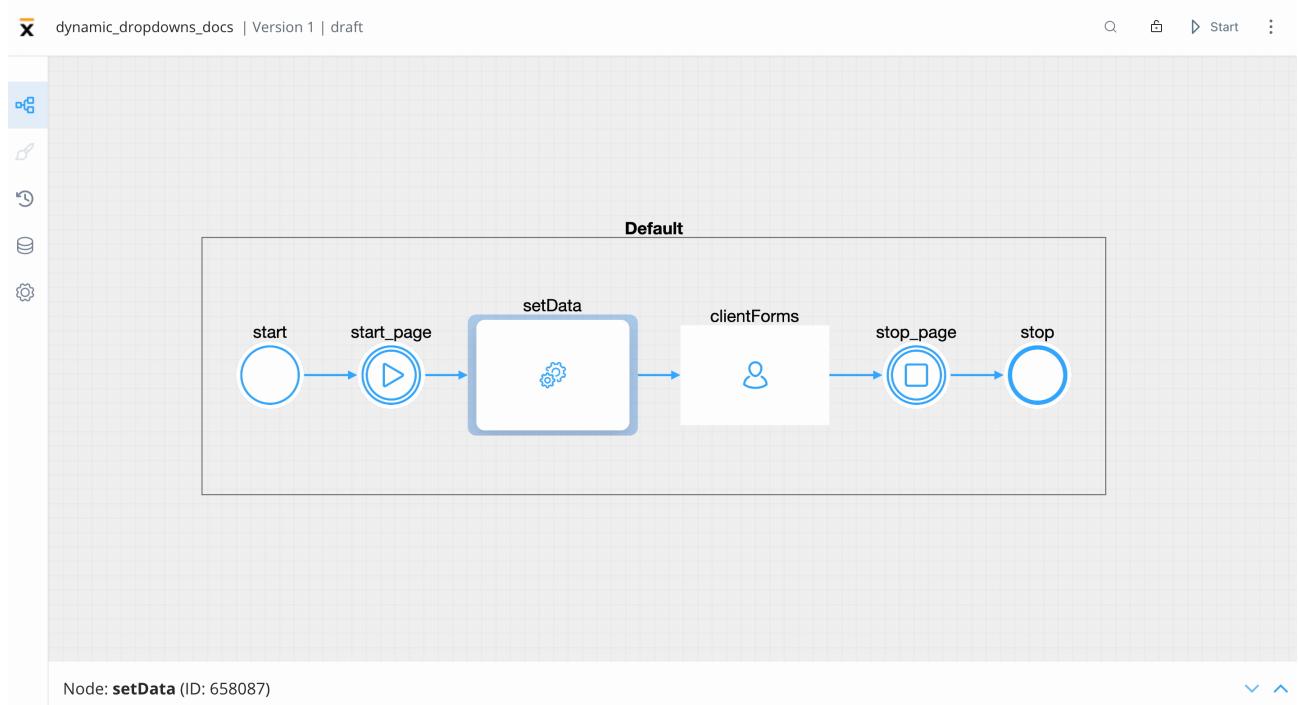
As mentioned previously, you can create dropdowns including static data, enumerations, or **process data**. Let's create an example using **process data** to create a process that contains **dynamic dropdowns**.

To create this kind of process, we need the following elements:

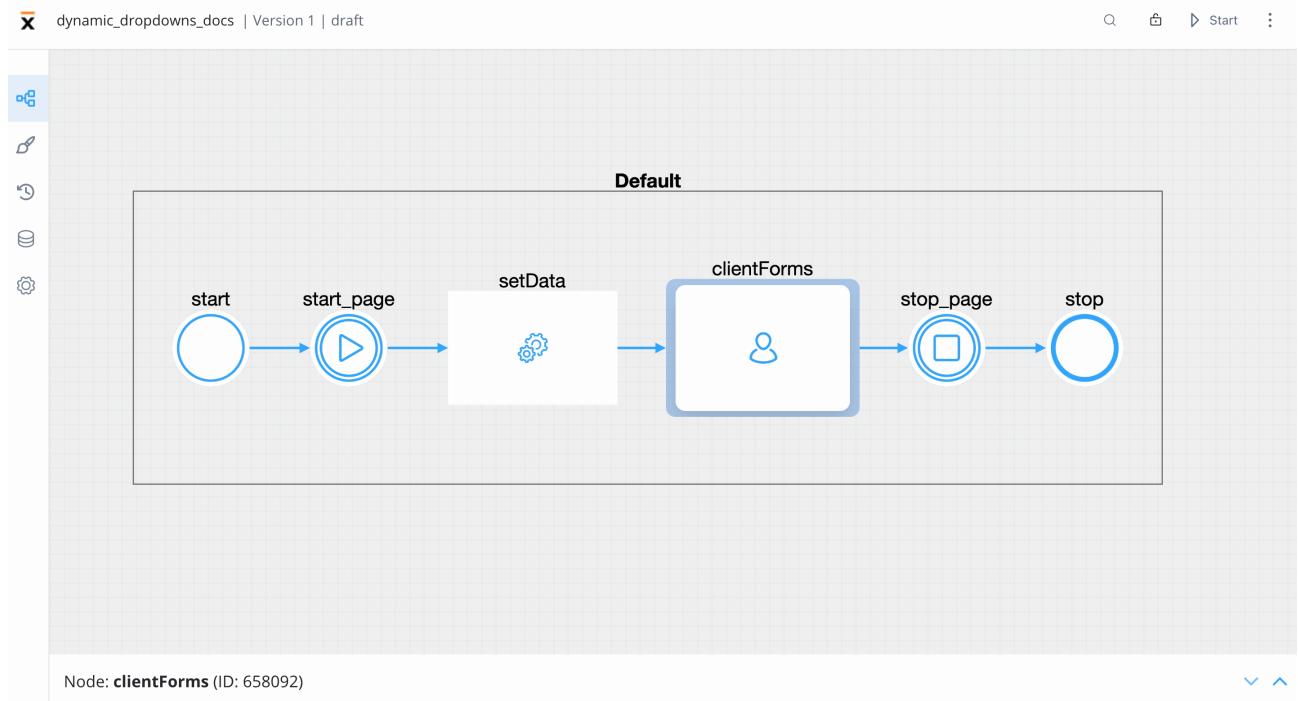
- a **start** node and an **end** node
- a **start milestone** UI element to it and an **end milestone** node



- a **task node** (this will be used to set which data will be displayed on the dropdowns)



- a **user task node** (here we have the client forms and here we add the SELECT elements)



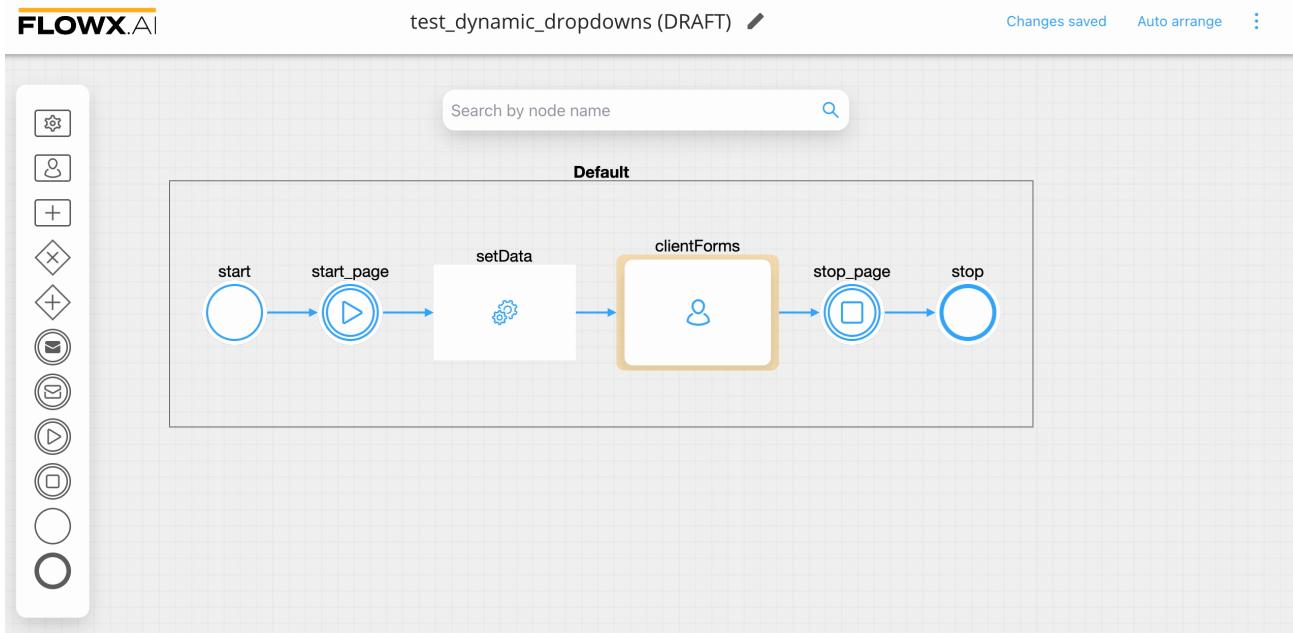
Creating the process

Follow the next steps to create the process from scratch:

1. Open **FLOWX Designer** and from the **Processes** tab select **Definitions**.
2. Click on the breadcrumbs (top-right corner) then click **New process** (the Process Designer will now open).
3. Now add all the **necessary nodes** (as mentioned above).

Configuring the nodes

1. On the **start milestone** node, add a **page** UI element.
2. On the **task node**, add a new **Action** (this will set the data for the dropdowns) with the following properties:
 - Action type - **Business Rule**
 - **Automatic**
 - **Mandatory**
 - **Language** (we used an **MVEL** script to create a list of objects)
3. On the **user task node**, add a new **Action** (submit action, this will validate the forms and save the date) with the following properties:
 - **Action type** - Save Data
 - **Manual**
 - **Mandatory**
 - **Data to send** (the key where the data will be sent) - **application**



Below you can find the MVEL script used in the above example:

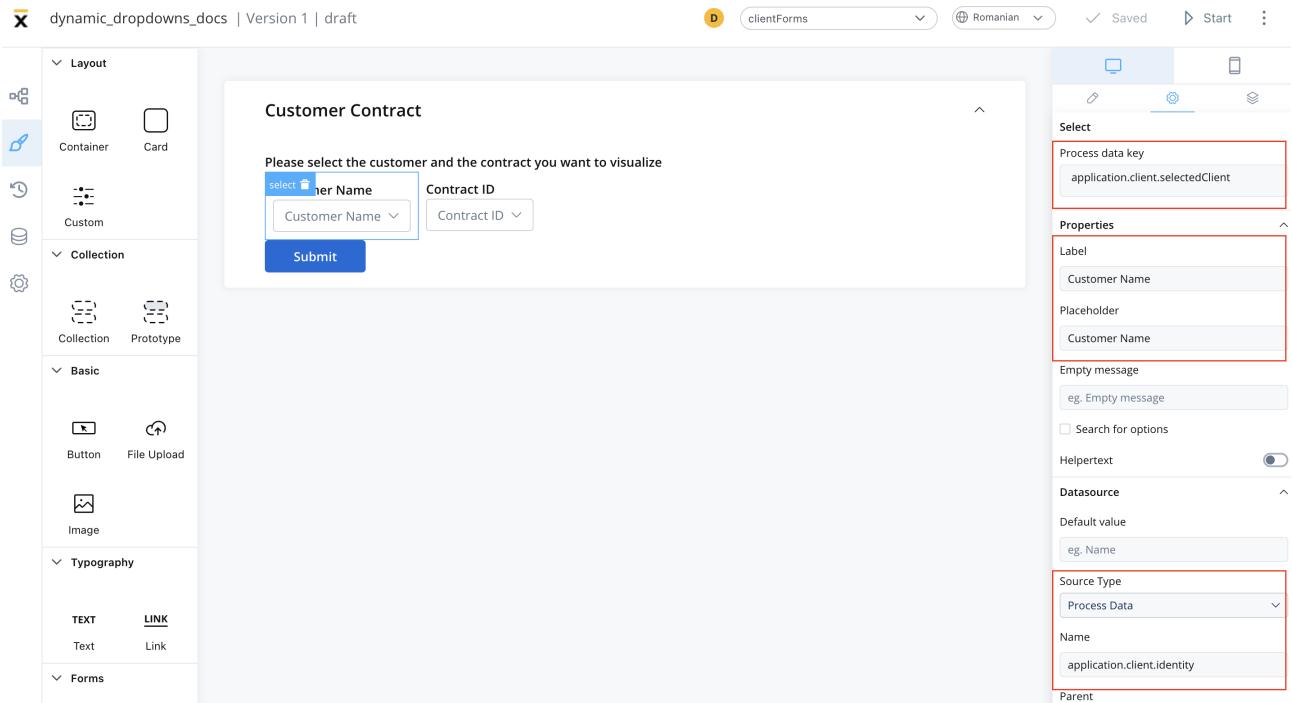
```
output.put("application",
{
    "client": {
        "identity": [
            {
                "value": "001",
                "label": "Eddard Stark"
            },
            {
                "value": "002",
                "label": "Sansa Stark"
            },
            {
                "value": "003",
                "label": "Catelyn Stark"
            }
        ]
    },
    "script": "output.put('application', { client: { identity: [ { value: '001', label: 'Eddard Stark' }, { value: '002', label: 'Sansa Stark' }, { value: '003', label: 'Catelyn Stark' } ] } })"
}
```

```
"contracts": {
    "001": [
        {
            "value": "c001",
            "label": "Eddard Contract 1"
        },
        {
            "value": "c007",
            "label": "Eddard Contract 2"
        }
    ],
    "003": [
        {
            "value": "c002",
            "label": "Catelyn Contract 1",
        },
        {
            "value": "c003",
            "label": "Catelyn Contract 2",
        },
        {
            "value": "c004",
            "label": "Catelyn Contract 3"
        }
    ],
    "002": [
        {
            "value": "c005",
            "label": "Sansa Contract 1",
        }
    ]
});
```

Configuring the UI

Follow the next steps to configure the UI needed:

1. Select the **user task node** and click the **brush icon** to open **UI Designer**
2. Add a **card** element as a **root component** (this will group the other elements inside it) with the following properties:
 - **Message** - `{"application": ${application}}`
 - **Title** - *Customer Contract*
3. Inside the **card**, add a **form element**.
4. Inside the **form** add two **select elements**, first will represent, for example, the *Customer Name* and the second the *Contract ID*.
5. For first select element (Customer Name) set the following properties:
 - **Process data key** - `application.client.selectedClient`
 - **Label** - Customer Name
 - **Placeholder** - Customer Name
 - **Source type** - Process Data (to extract the data added in the **task node**)
 - **Name** - `application.client.identity`



6. For the second select element (Contract ID) set the following properties:

- o **Process data key** - `application.client.selectedContract`
- o **Label** - Contract ID
- o **Placeholder** - Contract ID
- o **Source Type** - Process Data
- o **Name** - `application.contracts`
- o **Parent** - `SELECT` (choose from the dropdown list)

The screenshot shows the FLOWX.AI platform's form builder interface. On the left, there's a sidebar with various UI component categories: Layout (Container, Card, Custom), Collection (Collection, Prototype), Basic (Button, File Upload, Image), and Typography (TEXT, LINK). The main workspace displays a form titled "Customer Contract" with instructions: "Please select the customer and the contract you want to visualize". It features two dropdown menus: "Customer Name" and "Contract ID", and a "Submit" button. To the right of the form, a configuration panel for the "Contract ID" dropdown is open. This panel includes sections for "Select", "Properties", "Datasource", "Default value", "Source Type", "Name", "Parent", "Validators", "Expressions", and "Hide". The "Select" and "Source Type" sections are highlighted with a red border.

7. Add a button under the form that contains the select elements with the following properties:

- **Label** - Submit
- **Add UI action** - add the submit action attached earlier to the user task node

dynamic_dropdowns_docs | Version 1 | draft

Customer Contract

Please select the customer and the contract you want to visualize

Customer Name Contract ID

button mit

Properties

Label: Submit

UI Action

Add action

Event: CLICK

Action Type: ACTION

Node Action Name: submit

Use a different name for UI action

Add custom body

Add form to validate

Dismiss process?

Show loader?

8. Test and run the process by clicking **Start process**.

dynamic_dropdowns_docs | Version 1 | draft

Default

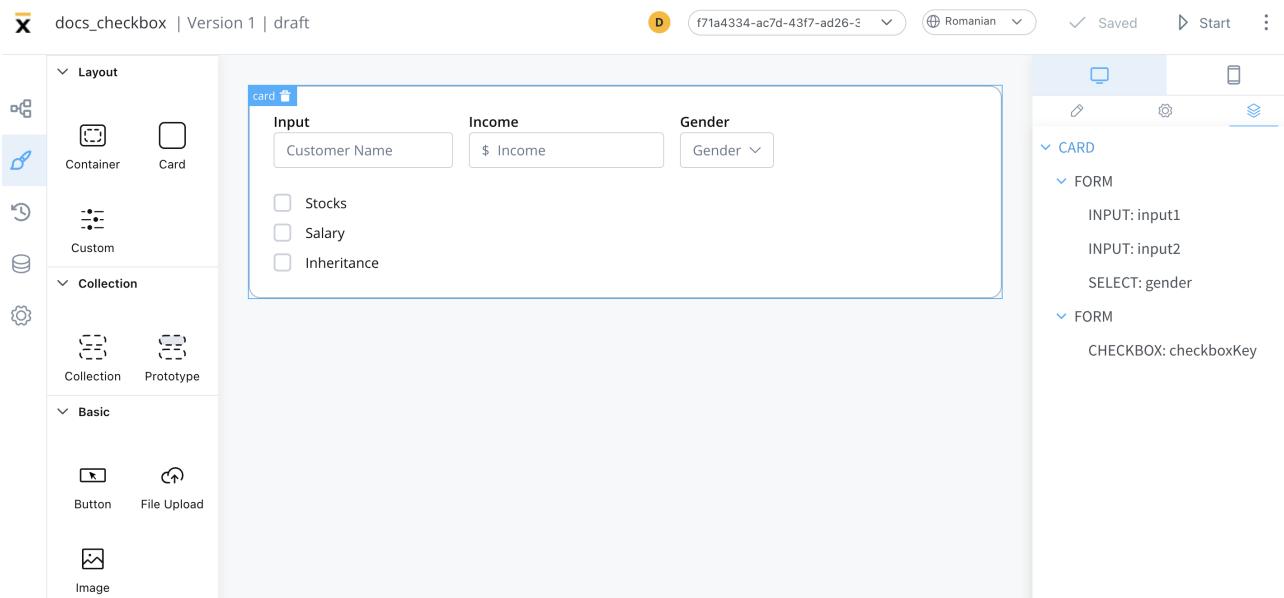
```

graph LR
    start((start)) --> startPage((start_page))
    startPage --> setData[setData]
    setData --> clientForms[clientForms]
    clientForms --> stopPage((stop_page))
    stopPage --> stop((stop))
  
```

Click any node to select

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Checkbox



A checkbox form field is an interactive element in a web form that provides users with multiple selectable options. It allows users to choose one or more options from a pre-determined set by simply checking the corresponding checkboxes.

This type of form field can be used to gather information such as interests, preferences, or approvals, and it provides a simple and intuitive way for users to interact with a form.

Configuring the checkbox element

Checkbox settings

The available configuration options for this form element are:

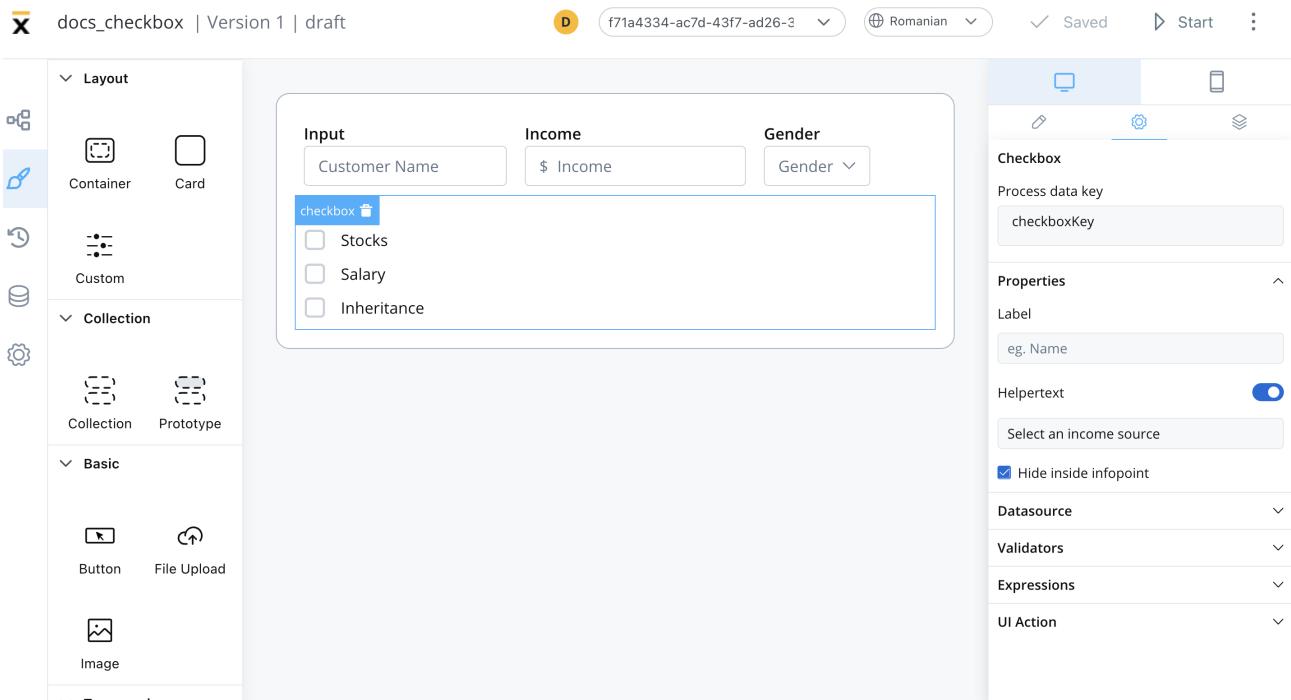
- [General](#)
- [Properties](#)
- [Datasource](#)
- [Validators](#)
- [Expressions](#)
- [UI actions](#)
- [Checkbox styling](#)

General

- **Process data key** - creates the binding between form element and process data, so it can be later used in [decisions](#), [business rules](#) or [integrations](#)

Properties

- **Label** - the label that appears on the checkbox
- **Helpertext** - additional information about the checkbox (can be hidden inside an infopoint)



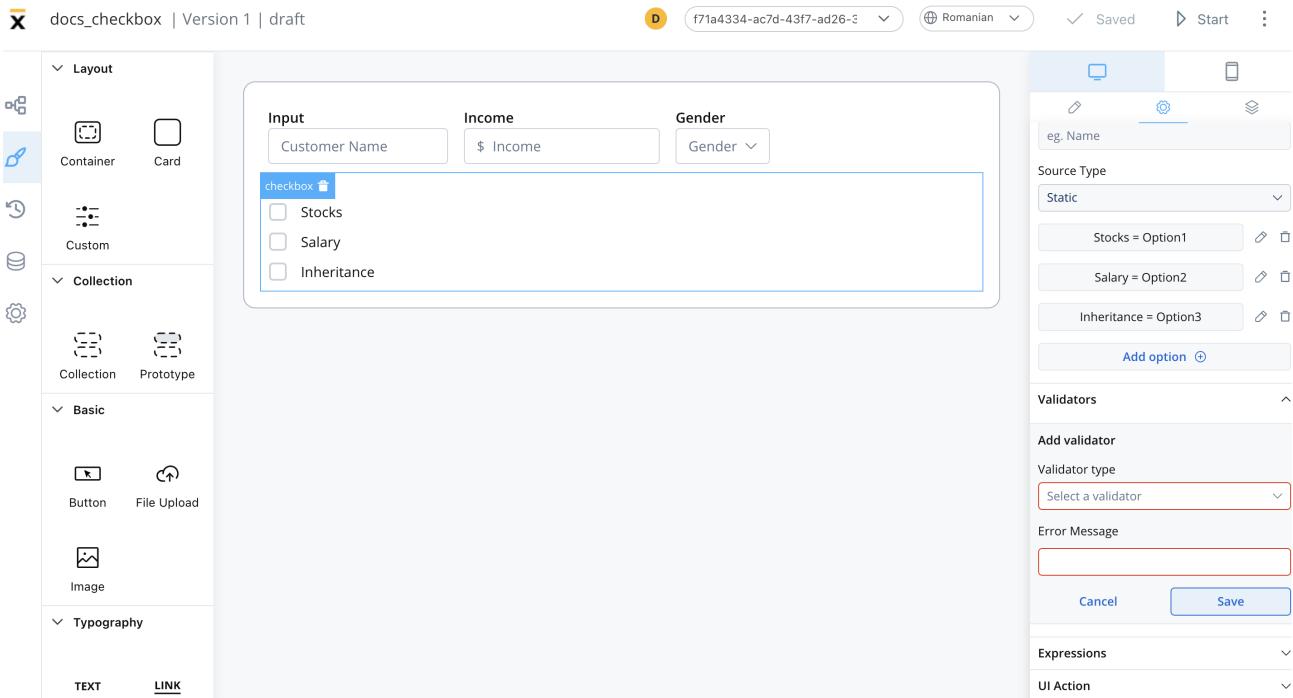
Datasource

- **Default Value** - the default value of the checkbox
- **Source Type** - it can be Static, Enumeration, or Process Data
- **Add option** - label - value pairs can be defined here

The screenshot shows the FLOWX.AI platform's form builder. On the left, there is a sidebar with various UI component icons categorized under 'Layout', 'Collection', 'Basic', and 'Custom'. The main workspace displays a form card with three input fields: 'Customer Name' (text), '\$ Income' (text), and 'Gender' (dropdown). Below these is a section titled 'checkbox' containing three options: 'Stocks', 'Salary', and 'Inheritance', each with a corresponding checkbox. To the right of the form is a preview area showing mobile and desktop versions of the page. The 'Properties' panel on the far right is expanded, showing configuration for the 'checkbox' component. A red box highlights the 'Datasource' section, which contains three static options: 'Stocks = Option1', 'Salary = Option2', and 'Inheritance = Option3'. There is also a 'Add option' button at the bottom of this section.

Validators

The following validators can be added to a checkbox: `required` and `custom` (more details [here](#)).



Expressions

The checkbox behavior can be defined using JavaScript expressions for hiding or disabling the element. The following properties can be configured for expressions:

- **Hide** - JavaScript expression used to hide the checkbox when it returns a truthy value
- **Disabled** - JavaScript expression used to disable the checkbox when it returns a truthy value

INFO

It's important to make sure that disabled fields have the same expression configured under the path expressions → hide.

UI actions

UI actions can be added to the checkbox element to define its behavior and interactions.

- **Event** - possible value: `CHANGE`
- **Action Type** - select the action type

! **INFO**

For more details on how to configure a UI action, click [here](#).

Checkbox styling

The type of the checkbox can be selected by using the **styling** tab in **UI Designer**, possible values:

- clear
- bordered

! **INFO**

For more valid CSS properties, click [here](#).

A clear checkbox element with three options added, and a column layout will look like as it follows:

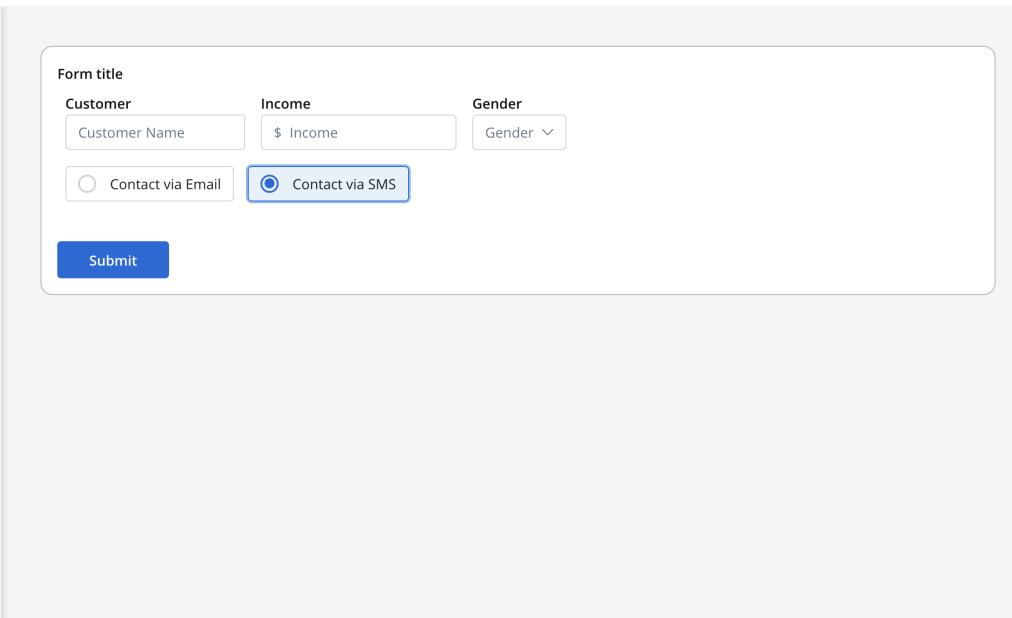
✓ First Form Title
First Form subtitle

Input 1 \$ Income Input 3 .000 Gender ▾

Stocks
 Salary
 Inheritance

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Radio



The screenshot shows the FLOWX.AI UI Designer interface. On the left, there's a sidebar with navigation links for Processes (Definitions, Active process, Process Instances, Failed process start) and Content Management (Enumerations, Substitution tags, Content models, Languages, Source systems, Media Library). The main area displays a form titled "Form title". The form contains three input fields: "Customer" (Customer Name), "Income" (\$ Income), and "Gender". Below these are two radio buttons: "Contact via Email" (unchecked) and "Contact via SMS" (checked). A blue "Submit" button is at the bottom.

Radio buttons are normally presented in radio groups (a collection of radio buttons describing a set of related options). Only one radio button in a group can be selected at the same time.

Configuring the radio field element

Radio settings

The available configuration options for this form element are:

- General
- Properties
- Datasource
- Validators
- Expressions
- UI actions
- Radio styling

General

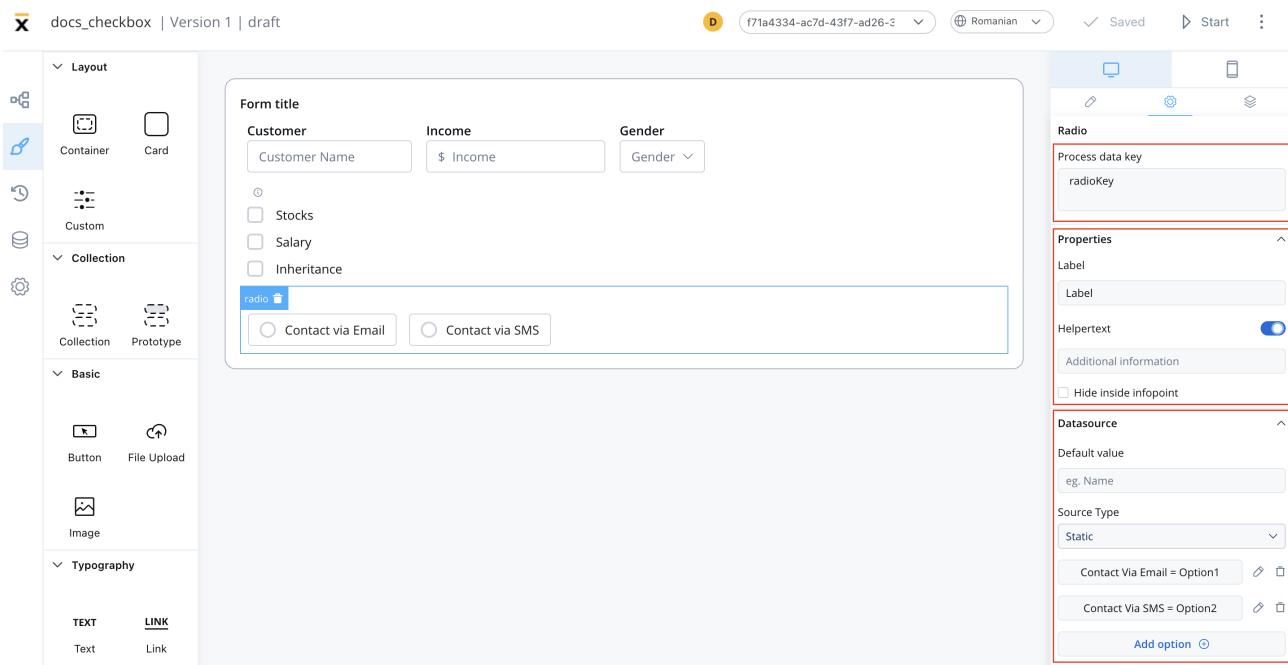
- **Process data key** - creates the binding between form element and process data so it can be later used in decisions, business rules or integrations

Properties

- **Label** - the label that appears on the radio
- **Helpertext** - additional information about the radio (can be hidden inside an infopoint)

Datasource

- **Default Value** - the default values of the radio element
- **Source Type** - it can be Static, Enumeration, or Process Data
- **Add option** - label - value pairs can be defined here



Validators

The following validators can be added to a radio: `required` and `custom` (more details [here](#))

Expressions

The radio's element behavior can be defined using JavaScript expressions for hiding or disabling the element. The following properties can be configured for expressions:

- **Hide** - JavaScript expression used to hide the Radio element when it returns a truthy value
- **Disabled** - JavaScript expression used to disable the Radio element when it returns a truthy value

!(INFO)

It's important to make sure that disabled fields have the same expression configured under the path expressions → hide.

The screenshot shows the FLOWX.AI builder interface. On the left, there is a sidebar with categories: Layout (Container, Card, Custom), Collection (Collection, Prototype), and Basic (Button, File Upload, Image). The main area displays a form titled "Form title". The form contains three input fields: "Customer" (Customer Name), "Income" (\$ Income), and "Gender" (Gender dropdown). Below these are three checkboxes: "Stocks", "Salary", and "Inheritance". A "radio" section follows, containing two radio buttons: "Contact via Email" and "Contact via SMS". On the right side, there is a panel for configuring the "Radio" component. It includes fields for "Process data key" (radioKey) and "Validators" (Required). The "Expressions" section is highlighted with a red border and contains fields for "Hide" and "Disabled", both currently empty. The top of the screen shows the project name "docs_checkbox | Version 1 | draft", a save status indicator, and language settings.

UI actions

UI actions can be added to the radio element to define its behavior and interactions.

- **Event** - possible value: CHANGE
- **Action Type** - select the action type

! INFO

For more details on how to configure a UI action, click [here](#).

Radio styling

The type of the radio can be selected by using the **styling** tab in **UI Designer**, possible values:

- clear
- bordered

! INFO

For more valid CSS properties, click [here](#).

A Radio element with two options added, and with a layout configuration set to horizontal will look like as it follows:

✓ First Form Title

First Form subtitle

The screenshot shows a UI Designer interface for a form titled "First Form Title". The subtitle is "First Form subtitle". The layout is set to "horizontal". There are three input fields: "Input 1", "Income" (with a dollar sign icon), and "Input 3". To the right of "Input 3" is a field "Gender" with a dropdown arrow. Below these are three radio buttons: "Stocks" (unchecked), "Salary" (checked), and "Inheritance" (unchecked). At the bottom are two buttons: "Contact via Email" (checked) and "Contact via SMS".

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Switch

Form title

Customer Income Gender

Customer Customer \$ 999999999 Gender

Contact via Email Contact via SMS

Select a date

09.02.1972

Do you want to subscribe to our newsletter?

Submit

A switch, a toggle switch, is another form element that can be utilized to create an intuitive user interface. The switch allows users to select a response by toggling it between two states. Based on the selection made by the user, the corresponding Boolean value of either true or false will be recorded and stored in the process instance values for future reference.

Configuring the switch element

Switch settings

The available configuration options for this form element are:

- General
- Properties
- Datasource
- Validators
- Expressions
- UI actions
- Switch styling

General

- **Process data key** - creates the binding between form element and process data so it can be later used in decisions, business rules or integrations

Properties

- **Label** - the label of the switch
- **Helpertext** - additional information about the switch element (can be hidden inside an infopoint)

Datasource

- **Default Value** - the default value of the switch form field (it can be switched on or switched off)

Validators

The following validators can be added to a switch element: `requiredTrue` and `custom` (more details [here](#)).

The screenshot shows the FLOWX.AI UI builder interface. On the left is a sidebar with categories like Layout, Collection, Basic, and Typography, each containing various UI components such as Container, Card, Custom, Collection, Prototype, Button, File Upload, and Image. The main area displays a form titled 'Form title' with fields for 'Customer Name', 'Income', 'Gender', and contact preferences ('Contact via Email' or 'Contact via SMS'). Below these is a date selector ('Select a date') and a switch element labeled 'want to subscribe to our newsletter?'. A 'Submit' button is at the bottom. To the right of the form is a properties panel with tabs for 'Properties', 'Validators', 'Expressions', and 'UI Action'. The 'Properties' tab is active, showing settings for the switch element, including a label ('Do you want to subscribe to ou'), helpertext, and a checked 'Default value' checkbox. The 'Validators' tab shows a 'requiredTrue' validator selected. The 'Expressions' and 'UI Action' tabs are also visible.

Expressions

- **Hide** - JavaScript expression used to hide the Switch element when it returns a truthy value
- **Disabled** - JavaScript expression used to disable the Switch element when it returns a truthy value

UI actions

UI actions can be added to the switch element to define its behavior and interactions.

- Event - possible value: CHANGE
- Action Type - select the action type

INFO

For more details on how to configure a UI action, click [here](#).

Switch styling

The label of the switch can be positioned either as `start` or `end`.

The screenshot shows the FLOWX.AI UI Designer interface. On the left, there's a sidebar with categories like Layout, Custom, Collection, Basic, and Media. In the center, a form titled "Form title" is displayed. It includes fields for "Customer Name", "Income", and "Gender". Below these are two radio buttons for "Contact via Email" and "Contact via SMS". A date picker for "Date of birth" is also present. At the bottom of the form is a switch component with the label "Do you want to subscribe to our newsletter?". To the right of the form is a panel titled "Switch" with properties for "Label position" set to "end", and other options like "Sizing", "Spacing", "Background", "Typography", and "Advanced".

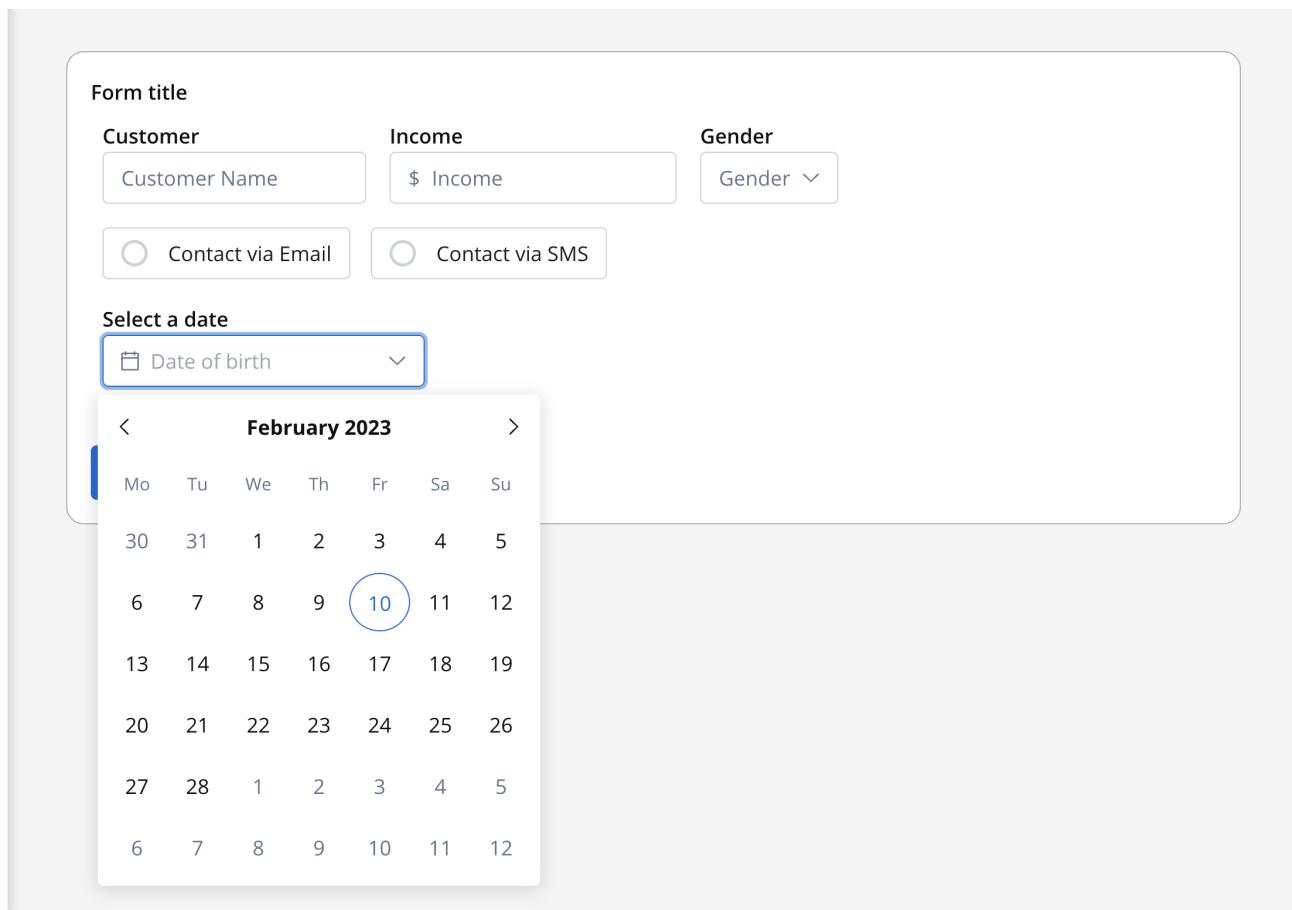
INFO

For more valid CSS properties, click [here](#).

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements /

Datepicker



The datepicker (Calendar Picker) is a lightweight component that allows end users to enter or select a date value.

!(INFO)

The default datepicker value is `DD.MM.YYYY`.

Configuring the datepicker element

Datepicker settings

The available configuration options for this form element are:

- [General](#)
- [Properties](#)
- [Datasource](#)
- [Validators](#)
- [Expressions](#)
- [UI actions](#)
- [Datepicker styling](#)

General

- **Process data key** - creates the binding between form element and process data so it can be later used in [decisions](#), [business rules](#) or [integrations](#)

Properties

- **Label** - the label of the datepicker
- **Placeholder** - placeholder when the field has no value
- **Min Date** - set the minimum valid date selectable in the datepicker
- **Max Date** - set the maximum valid date selectable in the datepicker
- **Min Date, Max Date error** - when a date is introduced by typing, define the error message to be displayed
- **Helpertext** - additional information about the input field (can be hidden inside an infopoint)

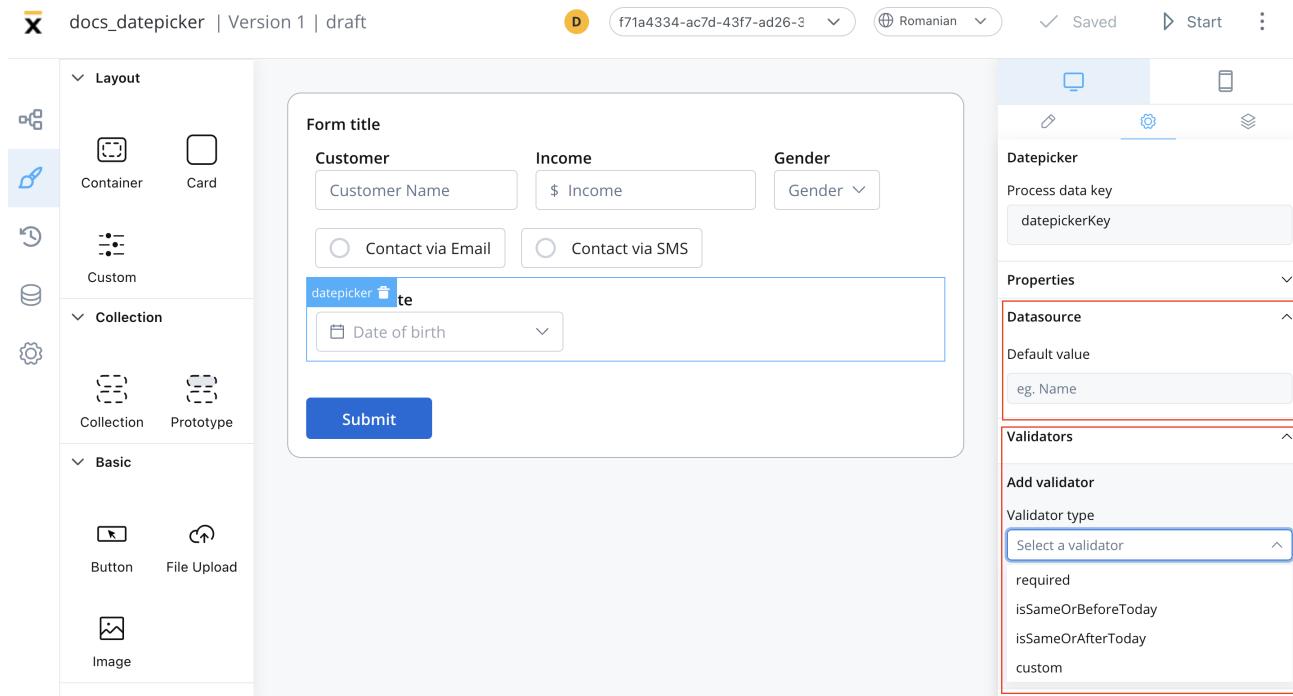
The screenshot shows the FLOWX.AI process builder interface. On the left, there's a sidebar with various building blocks categorized under Layout, Collection, and Basic. In the center, a form titled "Form title" is displayed. It contains fields for "Customer Name", "Income", "Gender", and two radio buttons for "Contact via Email" or "Contact via SMS". Below these is a "datepicker" element with a placeholder "Date of birth". A "Submit" button is at the bottom. To the right of the form is a properties panel for the "datepicker" element. The properties panel includes sections for "Process data key" (set to "datepickerKey"), "Properties" (with "Label" set to "Select a date" and "Placeholder" set to "Date of birth"), and validation settings for "Min date" and "Max date" (both set to "dd.mm.yyyy"). There are also sections for "Min date error" and "Max date error". A "Helpertext" section is shown with a toggle switch. The entire properties panel is highlighted with a red border.

Datasource

- **Default Value** - the default values of the datepicker element, this will autofocus the datepicker when you will run the process

Validators

The following validators can be added to a datepicker: `required`, `custom`, `isSameOrBeforeToday` or `isSameOrAfterToday` (more details [here](#)).



Expressions

The datepicker behavior can be defined using JavaScript expressions for hiding or disabling the element. The following properties can be configured for expressions:

- **Hide** - JavaScript expression used to hide the datepicker when it returns a truthy value
- **Disabled** - JavaScript expression used to disable the datepicker when it returns a truthy value

INFO

It's important to make sure that disabled fields have the same expression configured under the path expressions → hide.

UI actions

UI actions can be added to the datepicker element to define its behavior and interactions.

- **Event** - possible value: `CHANGE`
- **Action Type** - select the action type

INFO

For more details on how to configure a UI action, click [here](#).

The screenshot shows the FLOWX.AI builder interface. On the left, there's a sidebar with categories like Layout, Collection, and Basic, each containing icons for Container, Card, Custom, Collection, Prototype, Button, File Upload, and Image. The main area displays a form titled "Form title". The form contains fields for "Customer Name", "Income", "Gender", and two radio buttons for "Contact via Email" and "Contact via SMS". Below these is a datepicker field labeled "Date of birth". At the bottom is a blue "Submit" button. To the right of the form is a configuration panel. It has tabs for Desktop, Mobile, and Settings. Under Expressions, there are sections for Hide and Disabled, both currently empty. Under UI Action, there's a "Remove UI action" button and a section for "Event" which is set to "CHANGE". There's also a dropdown for "Action Type" with the placeholder "Select an action type". At the bottom of the configuration panel are "Cancel" and "Save" buttons.

Datepicker styling

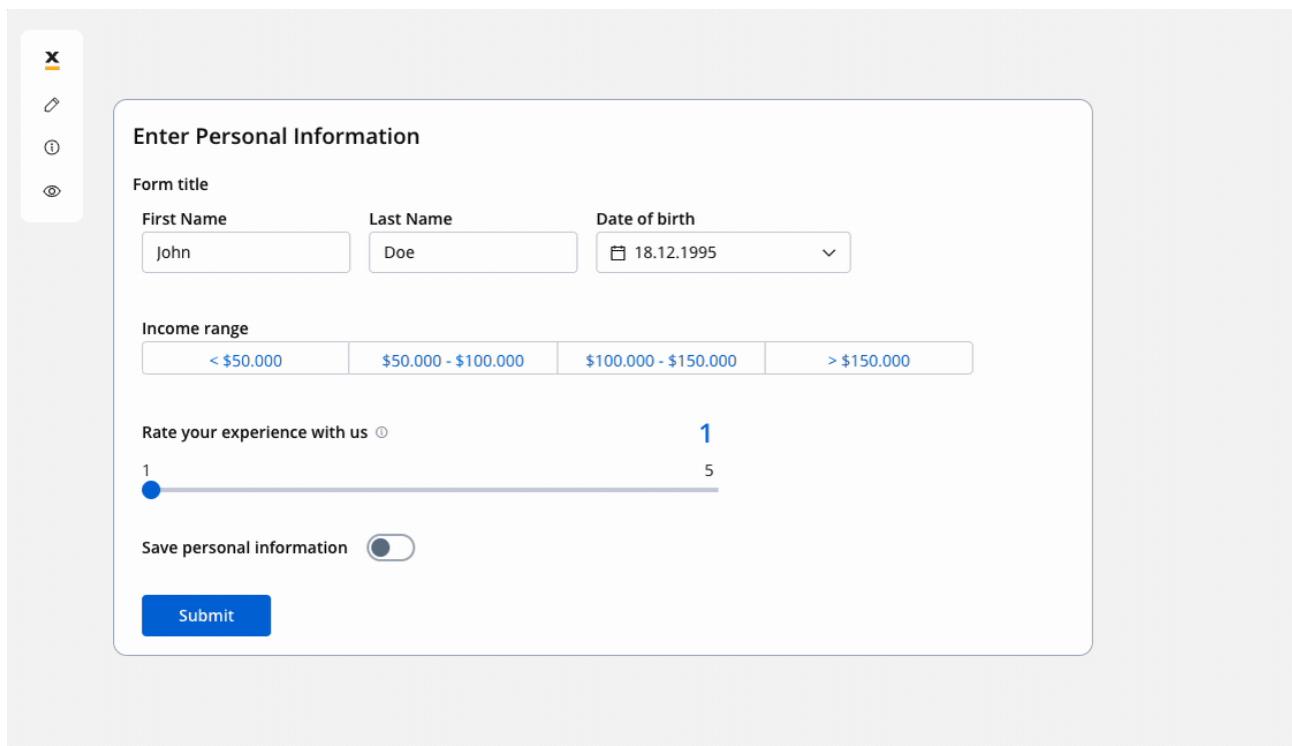
The styling of a datepicker element can be customized in various ways using CSS properties like typography color, border-radius/width, or advanced CSS params. This allows you to create a datepicker that fits seamlessly with the overall design of the application you are developing.

! INFO

For more valid CSS properties, click here.

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Slider



The screenshot shows a user interface for entering personal information. On the left, there's a toolbar with icons for close, edit, info, and preview. The main area has a title "Enter Personal Information". It includes fields for "First Name" (John), "Last Name" (Doe), and "Date of birth" (18.12.1995). Below these are segmented buttons for "Income range": < \$50.000, \$50.000 - \$100.000, \$100.000 - \$150.000, and > \$150.000. A slider titled "Rate your experience with us" ranges from 1 to 5, with the value set to 1. A toggle switch labeled "Save personal information" is turned on. At the bottom is a blue "Submit" button.

It allows users to pick only one option from a group of options, and you can choose to have between 2 and 5 options in the group. The segmented button is

easy to use, and can help make your application easier for people to use.

Configuring the slider element

Slider settings

The available configuration options for this form element are:

- **General**
- **Properties**
- **Datasource**
- **Validators**
- **Expressions**
- **UI actions**
- **Slider styling**

General

- **Process data key** - creates the binding between form element and process data so it can be later used in **decisions**, **business rules** or **integrations**

Properties

- **Label** - the label of the slider
- **Show value label** - a toggle option that determines whether the current selected value of the slider is displayed as a label alongside the slider handle
- **Helptext** - an optional field that provides additional information or guidance related to the usage or function of the slider, it can be hidden within an

infopoint, which users can expand or access for further detail

- **Min Value** - the minimum value or starting point of the slider's range, it defines the lowest selectable value on the slider
- **Max Value** - the maximum value or end point of the slider's range, it sets the highest selectable value on the slider
- **Suffix** - an optional text or symbol that can be added after the displayed value on the slider handle or value label, it is commonly used to provide context or units of measurement
- **Step size** - the increment or granularity by which the slider value changes when moved, it defines the specific intervals or steps at which the slider can be adjusted, allowing users to make more precise or discrete value selections

Datasource

- **Default Value** - the default value of the slider (static value - integer) the initial value set on the slider when it is first displayed or loaded, it represents a static value (integer), that serves as the starting point or pre-selected value for the slider, users can choose to keep the default value or adjust it as desired

The screenshot shows the FLOWX.AI builder interface. On the left, there's a sidebar with various UI components like Layout, Forms, and Collection. In the center, there's a form titled "Enter Personal Information" with fields for First Name, Last Name, Date of birth, Employment type (radio buttons for Employed and Pensioner), Income range (radio buttons for < \$50,000, \$50,000 - \$100,000, \$100,000 - \$150,000, and > \$150,000), and a Slider for Loan amount. The slider has a value of 55000 €, a minimum value of 10000 €, a maximum value of 100000 €, and a step size of 5000. A "Save personal information" toggle switch is also present. At the bottom is a "Submit" button. On the right, there's a configuration panel for the Slider component, showing the process data key "sliderKey" and various general properties such as Slider label, Helpertext, Min Value, Max Value, Suffix, and Step size.

Validators

The following validators can be added to a slider: `required` and `custom` (more details [here](#)).

UI actions

UI actions can be added to the slider element to define its behavior and interactions.

- **Event** - possible value: CHANGE
- **Action Type** - select the action type, ! for more details on how to configure a UI action, click [here](#)

Multiple sliders

You can also use multiple sliders UI elements that are interdependent, as you can see in the following example:

The screenshot shows a user interface for entering personal information. On the left, there's a sidebar with icons for close, edit, info, and refresh. The main area has a title 'Enter Personal Information'. It includes fields for First Name (placeholder 'First Name'), Last Name ('Doe'), and Date of birth ('18.12.1995'). Below that is a section for Employment type with radio buttons for 'Employed' and 'Pensioner'. There's a toggle switch for 'Save personal information'. The next section is 'Loan amount', which has a slider from 10000 \$ to 500000 \$. The current value is set to 255000 \$. To the right of the slider is a text input field containing '255000 \$'. Below this is another slider for 'Down payment' from 38250 \$ to 89250 \$. The current value is set to 38250 \$. To the right of this slider is a text input field containing '38250 \$'. At the bottom, there's a dropdown for 'Loan type' with 'Conventional' selected, and a blue 'Submit' button.

INFO

You can improve the configuration of the slider using computed values as in the example above. These values provide a more flexible and powerful approach for handling complex use cases. You can find an example by referring to the following documentation:

Dynamic & computed values

Slider styling

To create a slider with specific styling, sizing, typography, and color settings, you can use the following configuration:

- **Sizing**
- **Typography**
- **Background**

Sizing

- set the width of the button - fill/fixed/auto

Typography

Choose an appropriate font family, size, and weight for the button label text.

- **Label Color** - set the color of the button label text
- **Min/Max values** - set the color of
- **Result** - set the color of the

Background

- **Selected BG** - set the background color of the button.
- **ComponentBg** - set the background color of the button.

The screenshot shows the FLOWX.AI UI Designer interface. On the left, there's a sidebar with categories like Layout, Forms, and Collection, each containing icons for different components. The main area displays a form titled "Enter Personal Information". This form includes fields for First Name, Last Name, Date of birth, Employment type (radio buttons for Employed and Pensioner), Income range (a segmented button with options < \$50.000, \$50.000 - \$100.000, \$100.000 - \$150.000, and > \$150.000), and a Loan amount field with a slider set to 55000 €. Below these are Save personal information and Submit buttons. To the right of the form is a detailed sidebar for the "Slider" component, showing settings for Sizing (Fit W: fill), Spacing (0 to 16), Typography (Label, Min/Max values, Result), Background (Selected BG, ComponentBg), and Advanced (Add class).

INFO

For more valid CSS properties, click [here](#).

Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Form elements / Segmented button

The screenshot shows a form titled "Enter Personal Information". On the left, there is a vertical toolbar with icons for close, edit, info, and preview. The main area contains fields for First Name (John), Last Name (Doe), Date of birth (18.12.1995), and an income range segmented button with options: < \$50.000, \$50.000 - \$100.000, \$100.000 - \$150.000, and > \$150.000. Below these is a "Save personal information" toggle switch (which is turned on) and a blue "Submit" button.

It allows users to pick only one option from a group of options, and you can choose to have between 2 and 5 options in the group. The segmented button is easy to use, and can help make your application easier for people to use.

Configuring the segmented button

Segmented button settings

The available configuration options for this form element are:

- General
- Properties
- Datasource
- Validators
- Expressions

- **UI actions**
- **Segmented button styling**

General

- **Process data key** - creates the binding between form element and process data so it can be later used in **decisions**, **business rules** or **integrations**

Properties

- **Label** - the label of the segmented button
- **Helpertext** - additional information about the segmented button (can be hidden inside an infopoint)

Datasource

- **Default Value** - the default value of the segmented button (it can be selected from one of the static source values)
- **Source Type** - it is by default Static
- **Add option** - value/label pairs can be defined here

Validators

The following validators can be added to a segmented button: **required** and **custom** (more details [here](#)).

The screenshot shows the FLOWX.AI builder interface. A form titled "Enter Personal Information" is displayed on the canvas. The "Segmented Button" element from the "Forms" category in the sidebar is currently selected. The "Properties" panel on the right shows the label "Income range" and a help text field. The "Datasource" panel lists static source options for different income ranges. The "Validators" panel has a placeholder for adding a validator. Three specific sections of the "Properties", "Datasource", and "Validators" panels are highlighted with red boxes.

UI actions

UI actions can be added to the segmented button element to define its behavior and interactions.

- **Event** - possible value: CHANGE
- **Action Type** - select the action type

INFO

For more details on how to configure a UI action, click [here](#).

Segmented button styling

To create a segmented button with specific styling, sizing, typography, and color settings, you can use the following configuration:

- **Sizing**
- **Typography**
- **Background**

Sizing

- set the width of the button - fill/fixed/auto

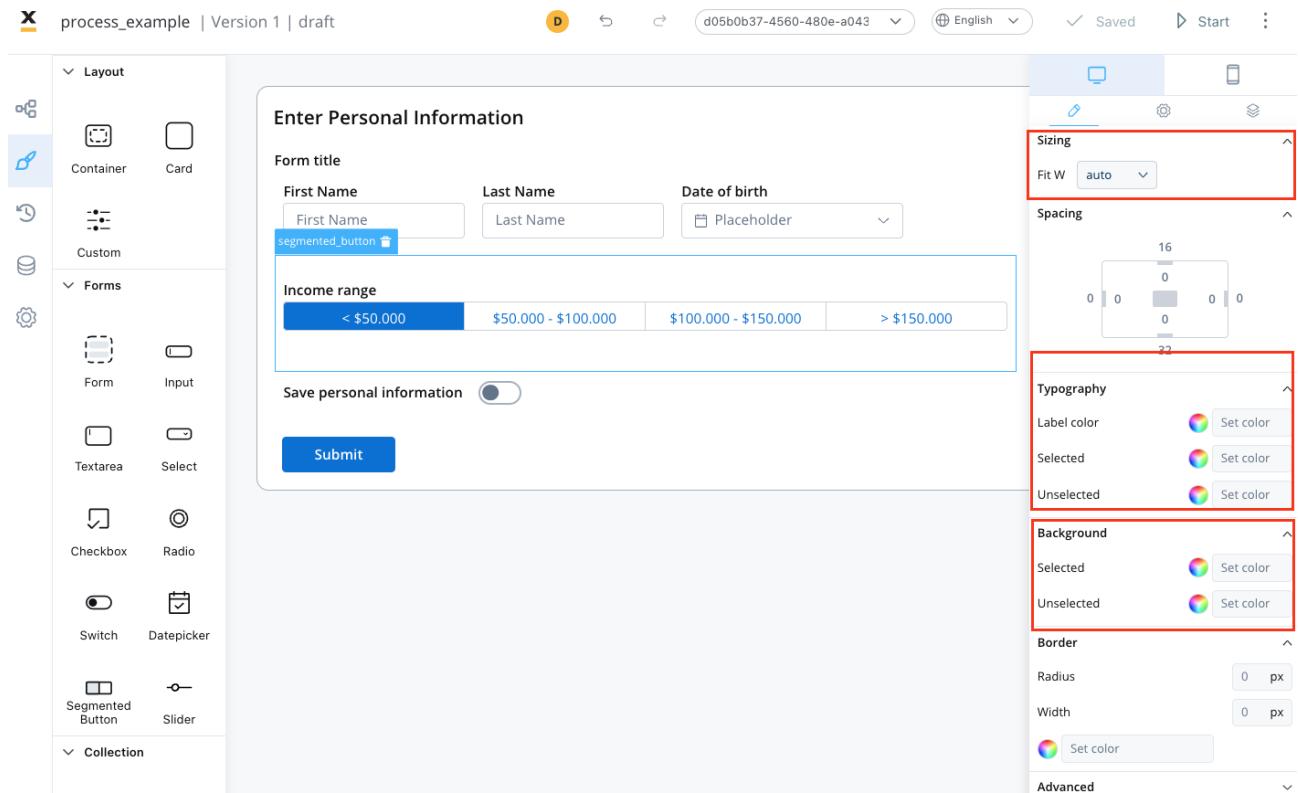
Typography

Choose an appropriate font family, size, and weight for the button label text.

- **Label Color** - set the color of the button label text
- **Selected State** - set the color of the label text when the button is selected
- **Unselected State** - set the color of the label text when the button is not selected

Background

- **Selected state** - set the background color of the button
- **Unselected state** - set the background color of the button



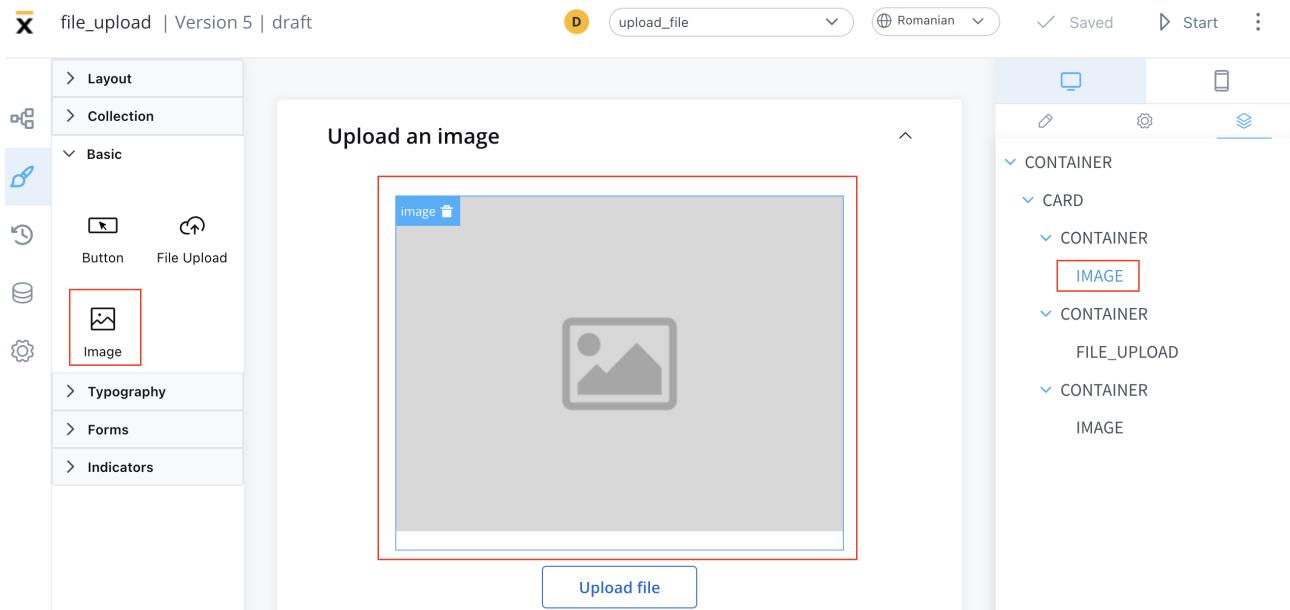
INFO

For more valid CSS properties, click [here](#).

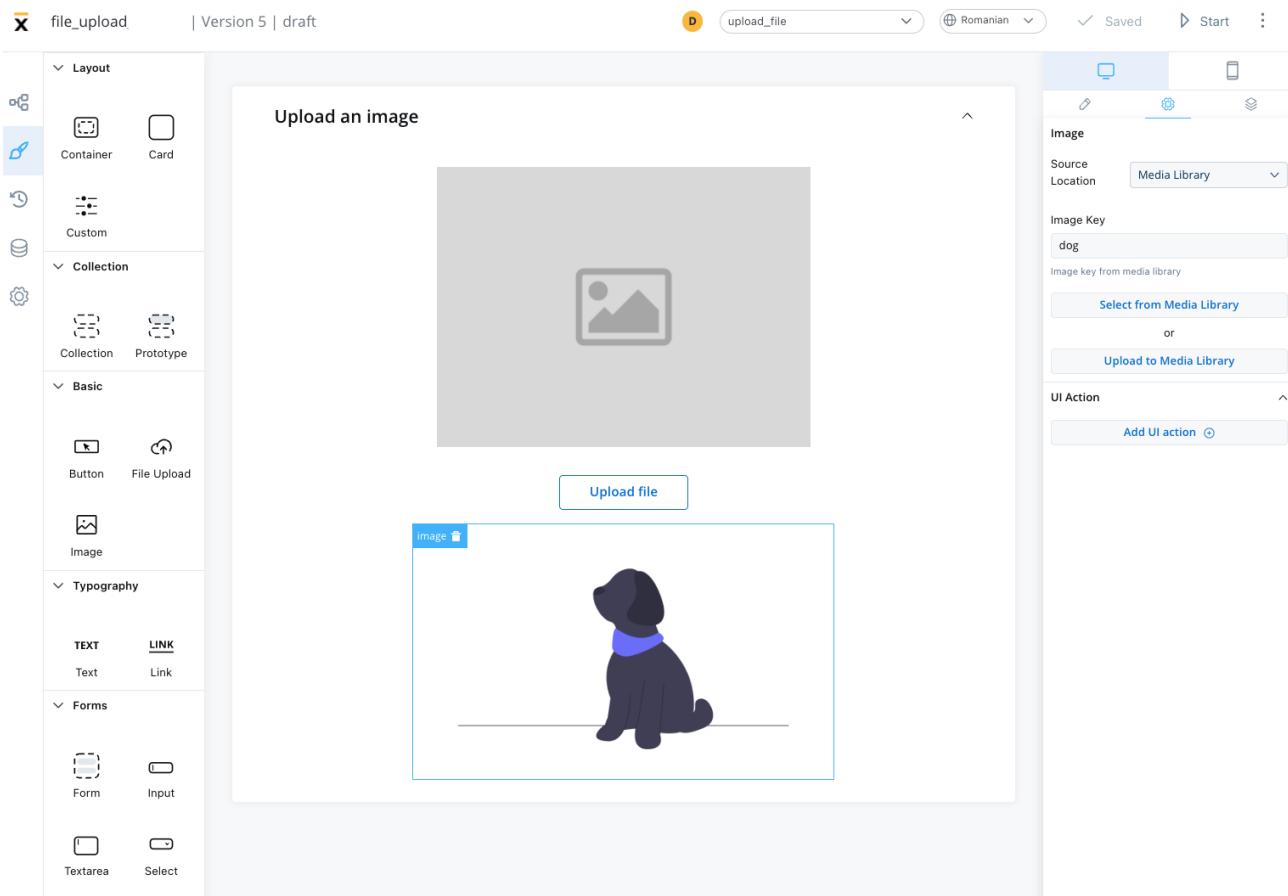
Was this page helpful?

BUILDING BLOCKS / UI Designer / UI component types / Image

Image UI elements are graphical components of a user interface that display a static or dynamic visual representation of an object, concept, or content.



These elements can be added to your interface using the UI Designer tool, and they are often used to convey information, enhance the aesthetic appeal of an interface, provide visual cues and feedback, support branding and marketing efforts, or present complex data or concepts in a more intuitive and accessible way.



Configuring an image

Configuring an image in the UI Designer involves specifying various settings and properties. Here are the key aspects of configuring an image:

Image settings

The image settings consist of the following properties:

- **Source location** - the location from where the image is loaded:
 - **Media Library**

- o **Process Data**
- o **External**

Depending on which **Source location** is selected, different configurations are available:

Media library

The screenshot shows the FLOWX.AI builder interface with the following details:

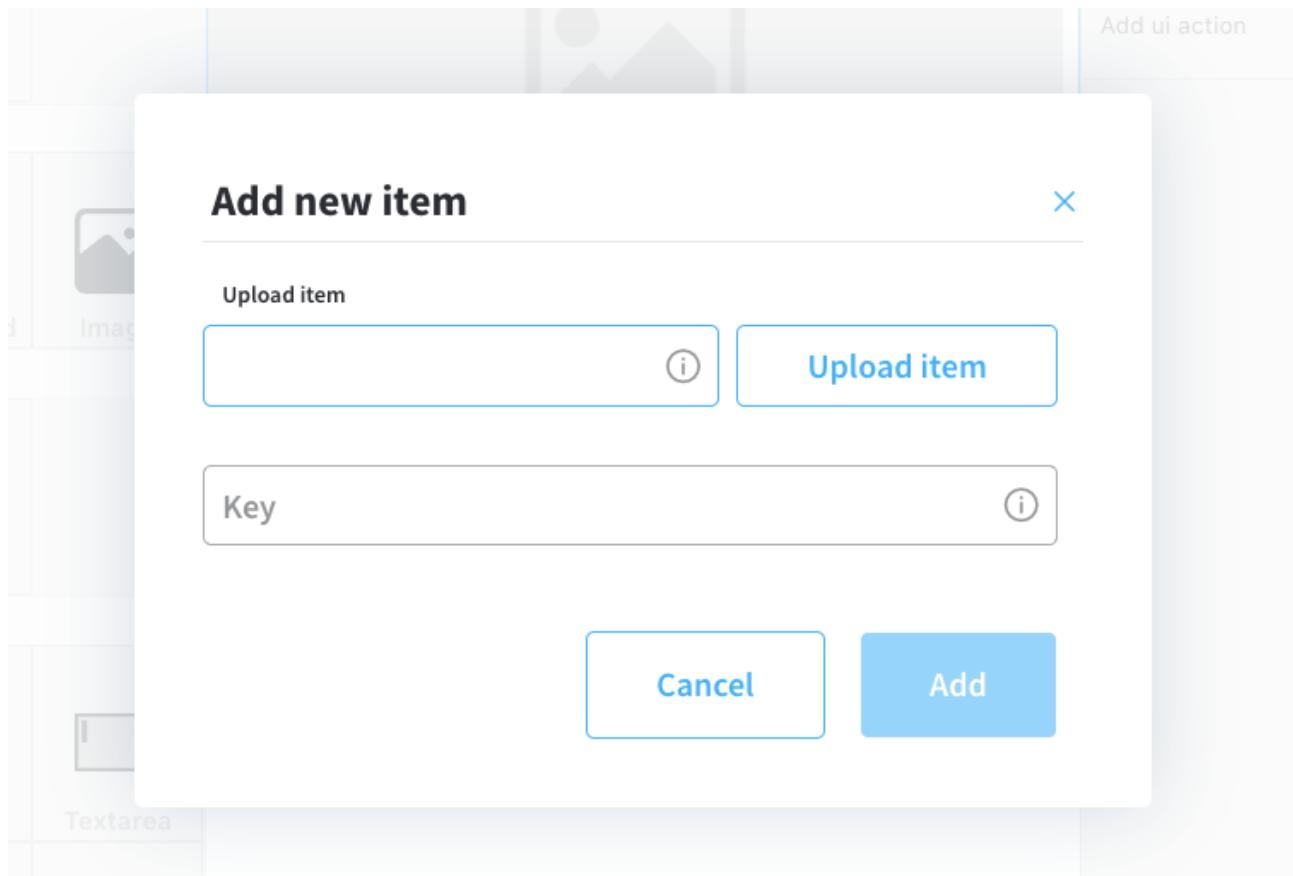
- Header:** file_upload | Version 5 | draft, upload_file, Romanian, Saved, Start, More options.
- Left Sidebar:** Layout (Container, Card), Collection (Custom, Collection, Prototype), Basic (Button, File Upload, Image), Typography (Text, Link), Forms.
- Middle Content Area:** A card titled "Upload an image" with a placeholder image icon and a "Upload file" button. Below it is a preview area showing a black dog sitting.
- Right Sidebar (highlighted with a red box):**
 - Image:** Source Location dropdown set to "Media Library".
 - Image Key:** Input field containing "dog".
 - UI Action:** "Add UI action" button.

- **Image key** - the key of the image from the media library

- **Select from media library** - search for an item by key and select it from the media library

Preview	Key	Format	Size	Edited at	Edited by
<input type="checkbox"/>	video_file	png	20.58 KB	23 Feb 2023, 11:29 AM	John Doe
<input type="checkbox"/>	switches	png	0.03 MB	23 Feb 2023, 11:28 AM	John Doe
<input type="checkbox"/>	form_example	png	22.29 KB	23 Feb 2023, 11:27 AM	John Doe
<input type="checkbox"/>	avatar2	png	0.03 MB	23 Feb 2023, 11:26 AM	John Doe
<input type="checkbox"/>	avatar1	png	27.73 KB	23 Feb 2023, 11:26 AM	John Doe
<input type="checkbox"/>	cat	png	15.63 KB	22 Feb 2023, 1:18 PM	John Doe
<input type="checkbox"/>	dog	png	20.65 KB	22 Feb 2023, 1:18 PM	John Doe

- **Upload to media library** - add a new item (upload an image on the spot)
 - **upload item** - supported formats: PNG, JPG, GIF, SVG, WebP;
! (maximum size - 1 MB)
 - **key** - the key must be unique and cannot be changed afterwards



Process Data

The screenshot shows the FLOWX.AI interface for building web applications. On the left, there is a sidebar with various UI component categories: Layout, Collection, Basic, Typography, and Forms. The 'Image' component is selected under the 'Basic' category. In the center, a modal window titled 'Upload an image' is displayed, featuring a placeholder image area with a 'Upload file' button below it. To the right of the modal, the component configuration panel is open, showing settings for 'Image'. These include 'Source Location' set to 'Process Data', 'Source Type' set to 'URL', 'Process Data Key' set to 'app.image', and a 'Placeholder Url' field containing 'Public url'. At the bottom of the configuration panel, there is a 'UI Action' section with a 'Add UI action' button.

- Identify the **Source Type**. It can be either a **URL** or a **Base 64 string**.
- Locate the data using the **Process Data Key**.
- If using a URL, provide a **Placeholder URL** for public access. This is the URL where the image placeholder is available.

file_upload | Version 5 | draft D upload_file Romanian Saved Start :

Layout

- Container
- Card

Custom

Collection

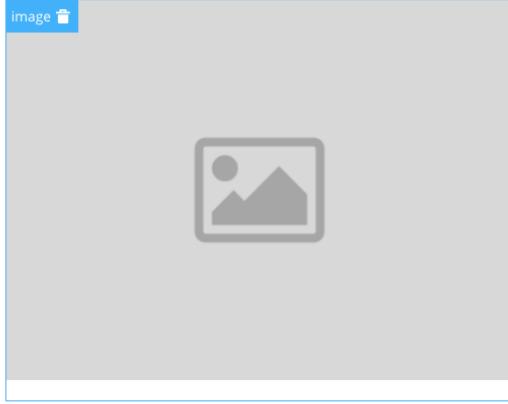
- Collection
- Prototype

Basic

- Button
- File Upload

Image

Upload an image



Upload file



Image

Source Location: Process Data

Source Type: URL

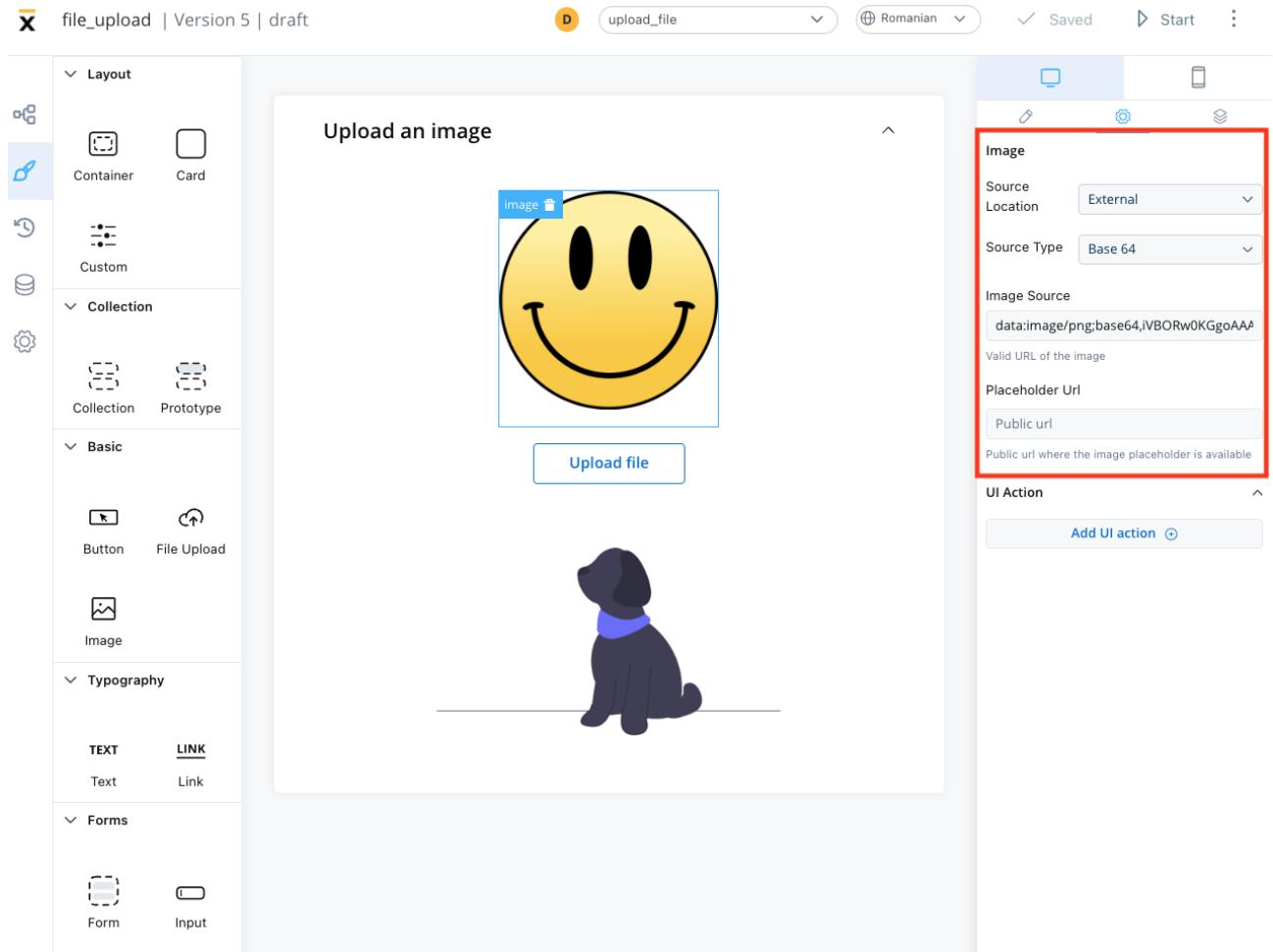
Process Data Key: app.image

Placeholder Url: Public url

UI Action

Add UI action

External



- **Source Type:** it can be either a **URL** or a **Base 64 string**
- **Image source:** the valid URL of the image.
- **Placeholder URL:** the public URL where the image placeholder is available

UI actions

The UI actions property allows you to add a UI Action, which must be configured on the same node. For more details on UI Actions, refer to the documentation [here](#).

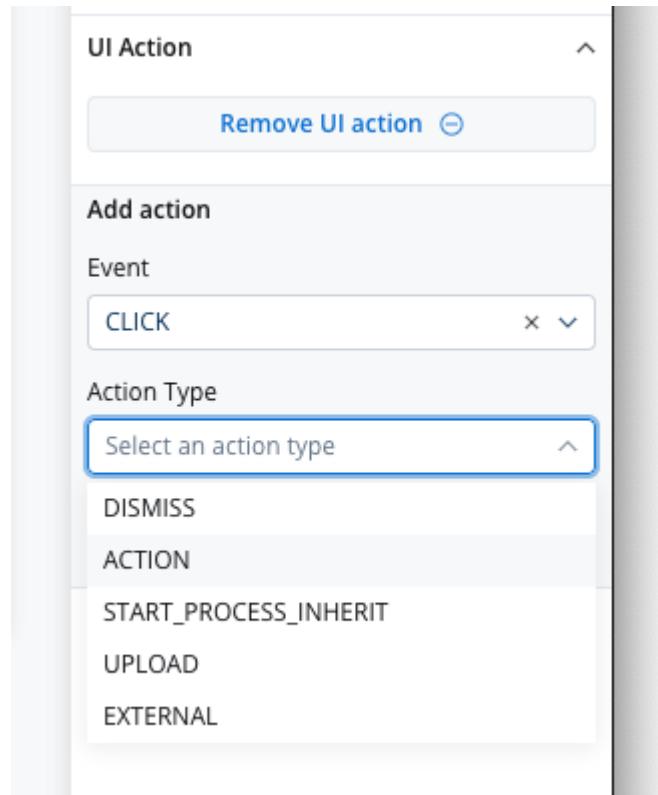
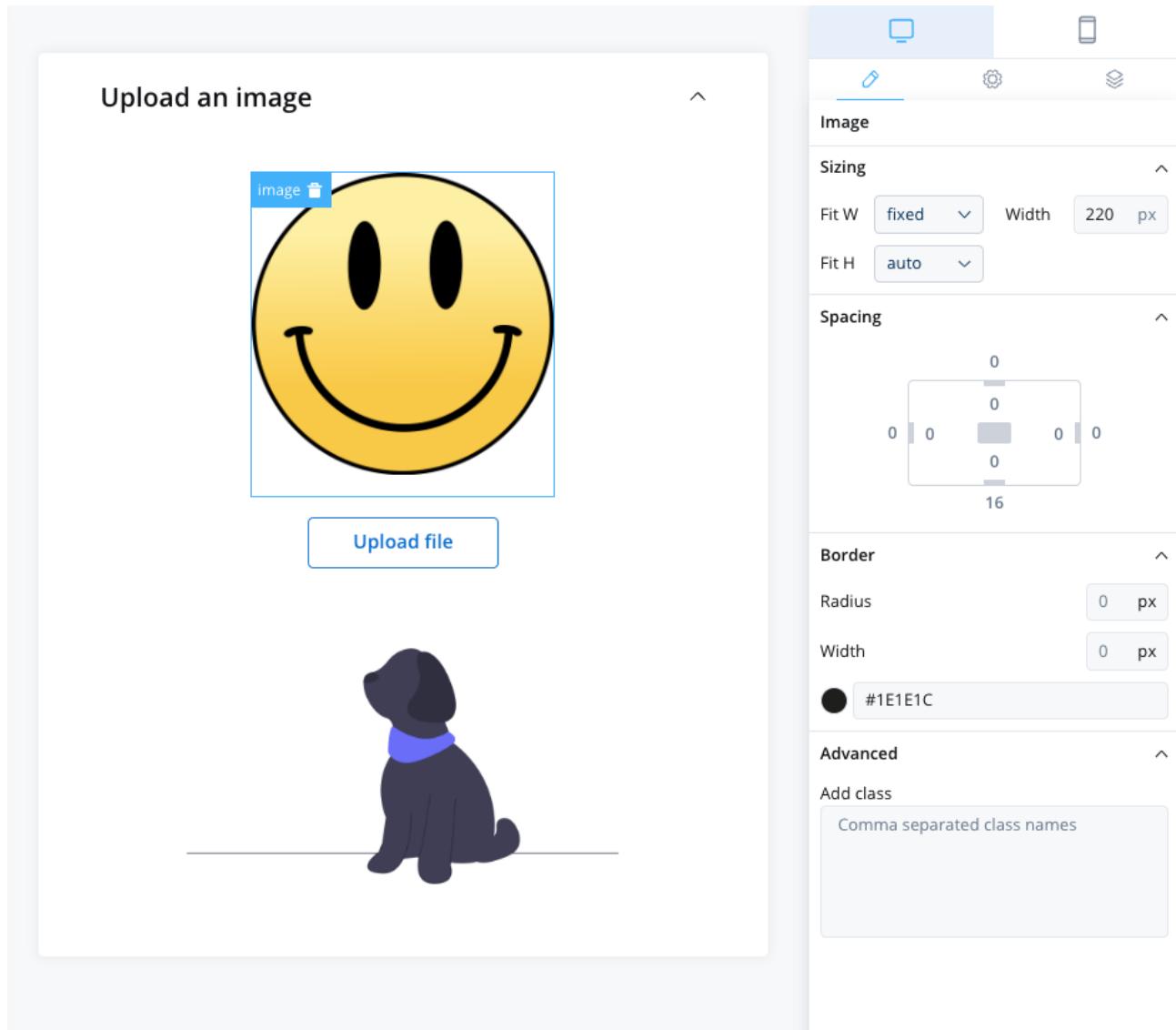


Image styling

The image styling property allows you to add or to specify valid CSS properties for the image. For more details on CSS properties, click [here](#).



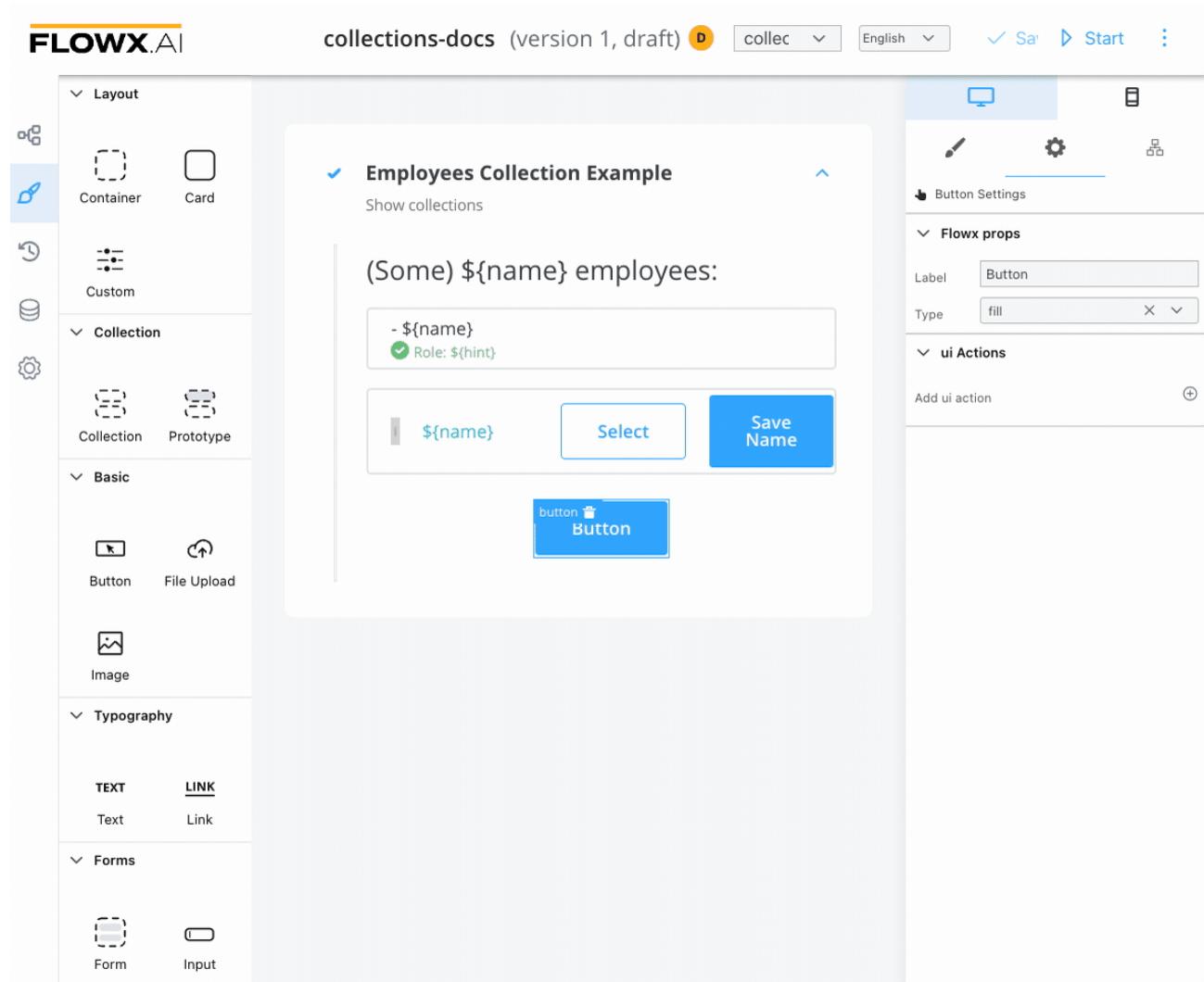
Was this page helpful?

BUILDING BLOCKS / UI Designer / UI actions

Multiple UI elements can be linked to an **action** via a UI Action. If the action is just a method to interact with the process, the UI Action adds information about how that UI should react. For example, should a loader appear after executing the action, should a modal be dismissed, or if some default data should be sent back to the process.

UI actions create a link between an **action** and a UI component or **custom component**.

The UI action informs the UI element to execute the given action when triggered. Other options are available for configuration when setting an action to a button and are detailed below.



There are two main types of UI Actions:

- Process UI Actions
- External UI Actions

Process UI actions

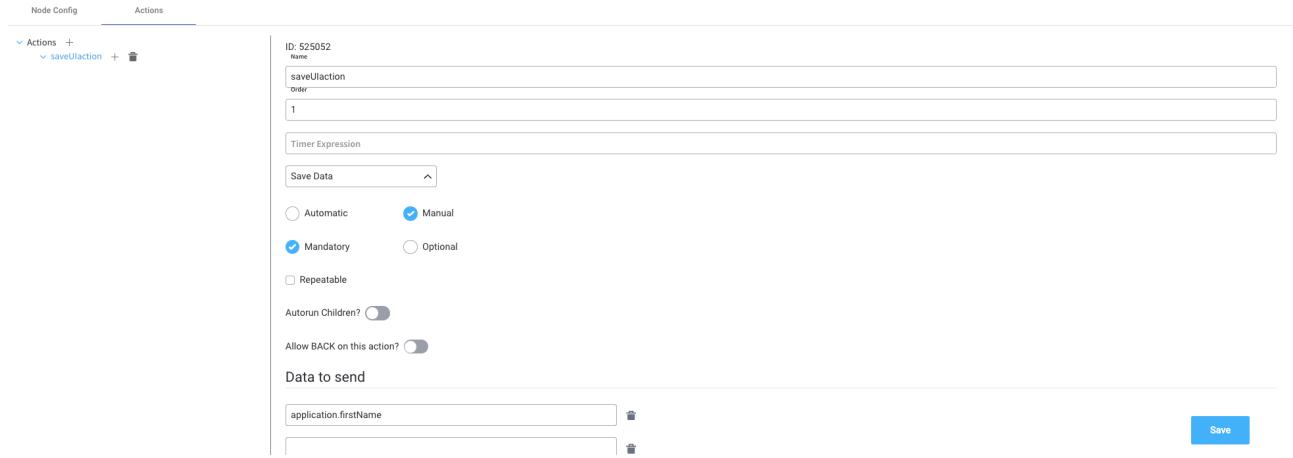
This is a UI action that describes how a **Button** (generated or custom) should interact with a process Manual action.

First, we need to configure the **manual action** that will be referred from the UI Action. For more information on how to add an action to a node, and how to configure it, check the following section:

» [Adding an action to a node](#)

Manual action configuration example - Save Data

1. Add an **action** to a node.
2. Select the action type - for example **Save Data**.
3. The action **type** should be **manual**.
4. **Keys** - it has two important implications:
 - First, this is a prefix of the keys that will send back by the UI Action link to this action. For example, if we have a big form with a lot of elements, but we need an action that just sends the email back (maybe creating email validation functionality) we will add just the key of that field:
`application.client.email`; if we need a button that will send back all the form elements that have keys that start with application.client we can add just this part
 - Second, a backend validation will be run to accept and persist just the data that start with this prefix. If we have three explicit keys,
`application.client.email`, `application.client.phone`,
`application.client.address` and we send
`application.client.age` this key will not be persisted



When this prerequisite is ready we can define the UI Action.

⚠ CAUTION

UI Actions and Actions should be configured on the same node.

UI action configuration example

Multiple configurations are available - **ACTION** type example:

- **Event**
- **Type**
- **Node Action Name** - dropdown with available actions for this node. If the dropdown is empty please add a manual action that is required before we create the UI Action
- **Use a different name for UI action**
- **UI action name - this becomes** important when the action is used in a **Custom component** as it will be used to trigger the action. For UI actions added on a generated button component this name is just descriptive

- **Custom body** - this is the default response in JSON format that will be merged with any extra parameters added explicitly when executing the action, by a web application (from a [custom component](#))
- **Forms to validate** - select from the dropdown what element will be validated (you can also select the children)
- **Dismiss process** - if the UI Actions is added on a subprocess and this parameter is true, triggering this UI action will dismiss the subprocess view (useful for modals subprocess)
- **Show loader?** - a loader will be displayed if this option is true until a web-socket event will be received (new screen or data)

ui Actions

Add ui action +

Add action

X

Event X ▾

CLICK

Type X ▾

ACTION

Node Action Name X ▾

saveUlation

Use a different name for UI action

Custom body

Forms to validate ▼

FORM

Dismiss process? Show loader?

Cancel Save

UI actions elements

Events

You can add an event depending on the element that you select. There are two events types available: **CLICK** and **CHANGE**.

The screenshot shows the FLOWX.AI UI builder interface. On the left, there is a sidebar with various UI element categories: Layout (Container, Card, Custom), Collection (Collection, Prototype), Basic (Button, File Upload, Image), Typography (Text, Link), and Forms (Form, Input). In the center, there is a card titled "First Form Title" with a subtitle "First Form Subtitle". Inside the card, there is a "Form title" section and three input fields labeled "Input 1", "Input 2", and "Input 3". Below the card, there is a blue button labeled "button" with the text "Button". On the right, there is a sidebar with "Flowx props" settings for a "Button" type element, showing "Label: Button" and "Type: fill". Below this, there is an "ui Actions" section with a button labeled "Add ui action".

INFO

! Not available for *UI Actions* on Custom Components.

Action types

The **action type** dropdown will be pre-filled with the following UI action types:

- DISMISS - used to dismiss a modal after action execution
- ACTION - used to link an action to a UI action
- START_PROCESS_INHERIT - used to inherit data from another process
- UPLOAD - used to create an un upload action
- EXTERNAL - used to create an action that will open a link in a new tab

External UI actions

Used to create an action that will open a link in a new tab.

If we toggle the EXTERNAL type, a few new options are available:

1. **URL** - web URL that will be used for the external action
2. **Open in new tab** - this option will be available to decide if we want to run the action in the current tab or open a new one

Add action

Type

EXTERNAL

Ui action Name

RedirectToGoogle

Url

www.google.com

Params

Dismiss process? Open in new tab?

[Cancel](#) [Save](#)

For more information on how to add actions and how to configure a UI, check the following section:

» [Managing a process flow](#)

[Was this page helpful?](#)

BUILDING BLOCKS / UI Designer / Validators

Validators are an essential part of building robust and reliable applications. They ensure that the data entered by the user is accurate, complete, and consistent. In Angular applications, validators provide a set of pre-defined validation rules that can be used to validate various form inputs such as text fields, number fields, email fields, date fields, and more.

The screenshot shows the FLOWX.AI interface for building a form. On the left, there's a sidebar with categories like Layout, Collection, Basic, Typography, and Forms, each containing icons for different components. The main area shows a form with a title, input fields, and a submit button. On the right, there's a panel for configuring form elements, including suffixes, help texts, data sources, default values, and validators. A red box highlights the 'Validators' section, which contains a dropdown menu titled 'Select a validator' and a list of options: required, minlength, maxlength, min, max, email, pattern, and custom.

Angular provides default validators such as:

1. **min**
2. **max**
3. **minLength**
4. **maxLength**
5. **required**
6. **email**

Other predefined validators are also available:

1. `isSameOrBeforeToday`: validates that a **datepicker** value is in the past
2. `isSameOrAfterToday`: validates that a datepicker value is in the future

 **INFO**

Form validation is triggered by default when the button set to validate a **form** is pressed.

It's also possible to build **custom validators** inside the container application and reference them here by name.

Predefined validators

required validator

This validator checks whether a value exists in the input field.

The screenshot shows the FLOWX.AI interface for building a form. On the left, there's a sidebar with categories like Layout, Collection, Basic, and Typography, each containing various UI components. In the center, a preview window shows a form with a title "Form title" and two inputs labeled "Input label". The right side is the configuration panel for the selected input. It includes fields for Type (text), Prefix (eg. prefix), Suffix (eg. suffix), Helpertext (with a toggle switch), Datasource, Default value (eg. Name), and Validators. The "Validators" section is highlighted with a red box and contains three items: "Required", "Minlength", and a button "Add a validator". Below it is the "Expressions" section with fields for Hide and Disabled.

It is recommended to use this validator with other validators like **minlength** to check if there is no value at all.

This screenshot shows the result of applying the validators. The "Input label" field now has a red border and a red placeholder "Placeholder". Below the input, the message "This input is required" is displayed in red. The "Submit" button is blue.

minlength validator

This validator checks whether the input value has a minimum number of characters. If there are no characters at all, this validator will not trigger. It is advisable to use this validator with a **required** validator.

The screenshot shows a modal dialog titled "Add validator". Under "Validator type", "minlength" is selected. In the "Params" field, the value "5" is entered. In the "Error Message" field, the message "At least 5 characters" is displayed. At the bottom, there are "Cancel" and "Save" buttons, with "Save" being highlighted.

maxlength validator

This validator checks whether the input value has a maximum number of characters. If there are no characters at all, this validator will not trigger. It is advisable to use this validator with a **required** validator.

Add validator

Validator type

maxlength

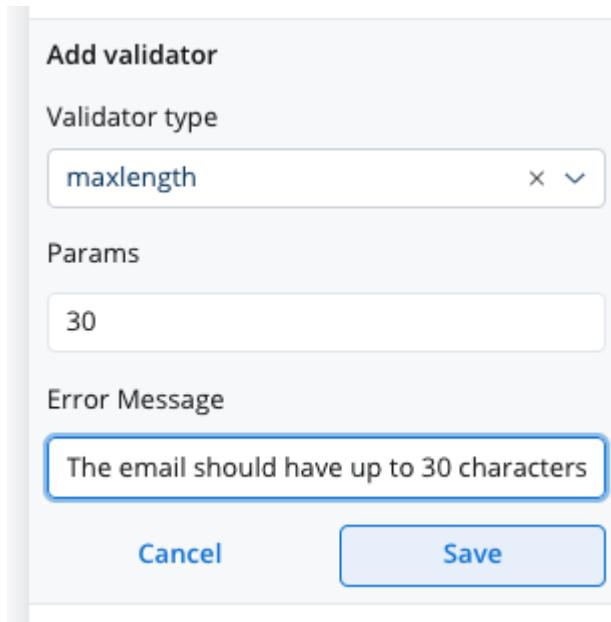
Params

30

Error Message

The email should have up to 30 characters

Cancel Save



min validator

This validator checks whether a numeric value is smaller than the specified value. If there are no characters at all, this validator will not trigger. It is advisable to use this validator with a [required](#) validator.

Edit validator

Validator type

min

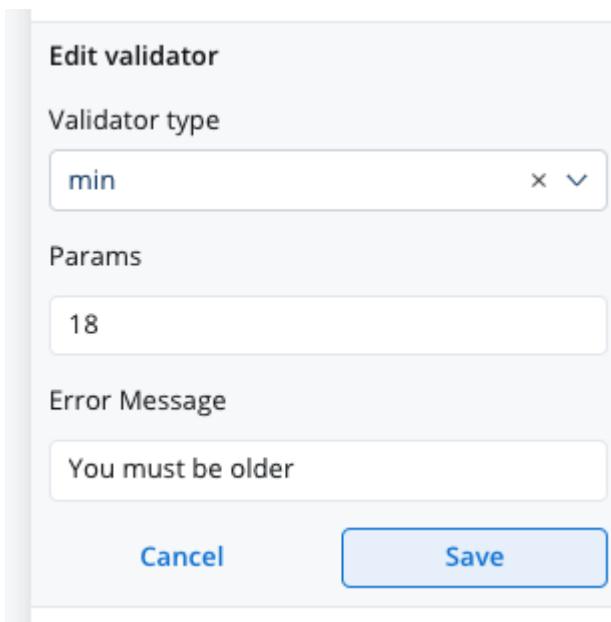
Params

18

Error Message

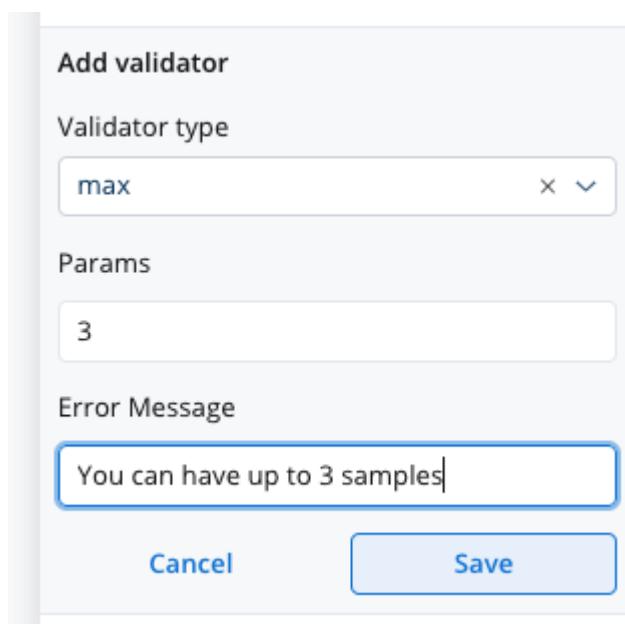
You must be older

Cancel Save



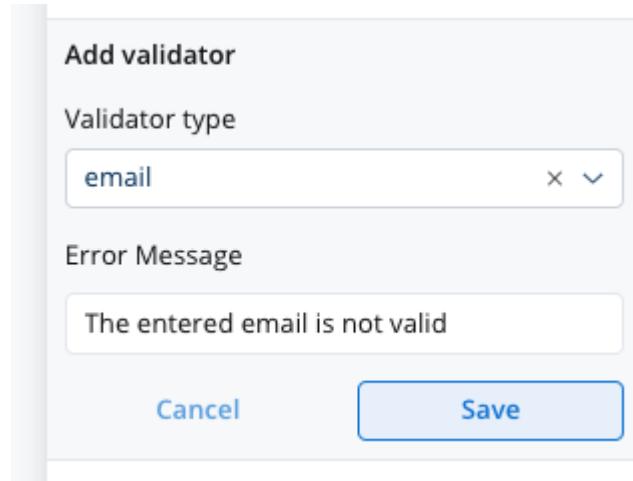
max validator

This validator checks whether a numeric value is larger than the specified value. If there are no characters at all, this validator will not trigger. It is advisable to use this validator with a [required](#) validator.



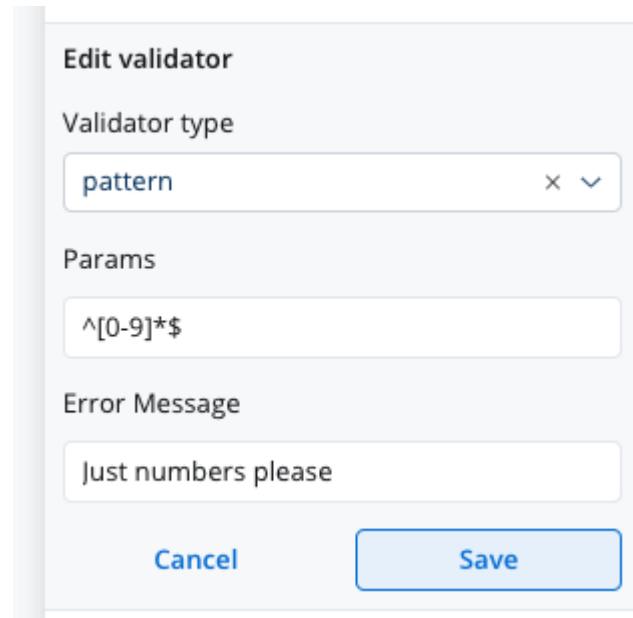
email validator

This validator checks whether the input value is a valid email. If there are no characters at all, this validator will not trigger. It is advisable to use this validator with a [required](#) validator.



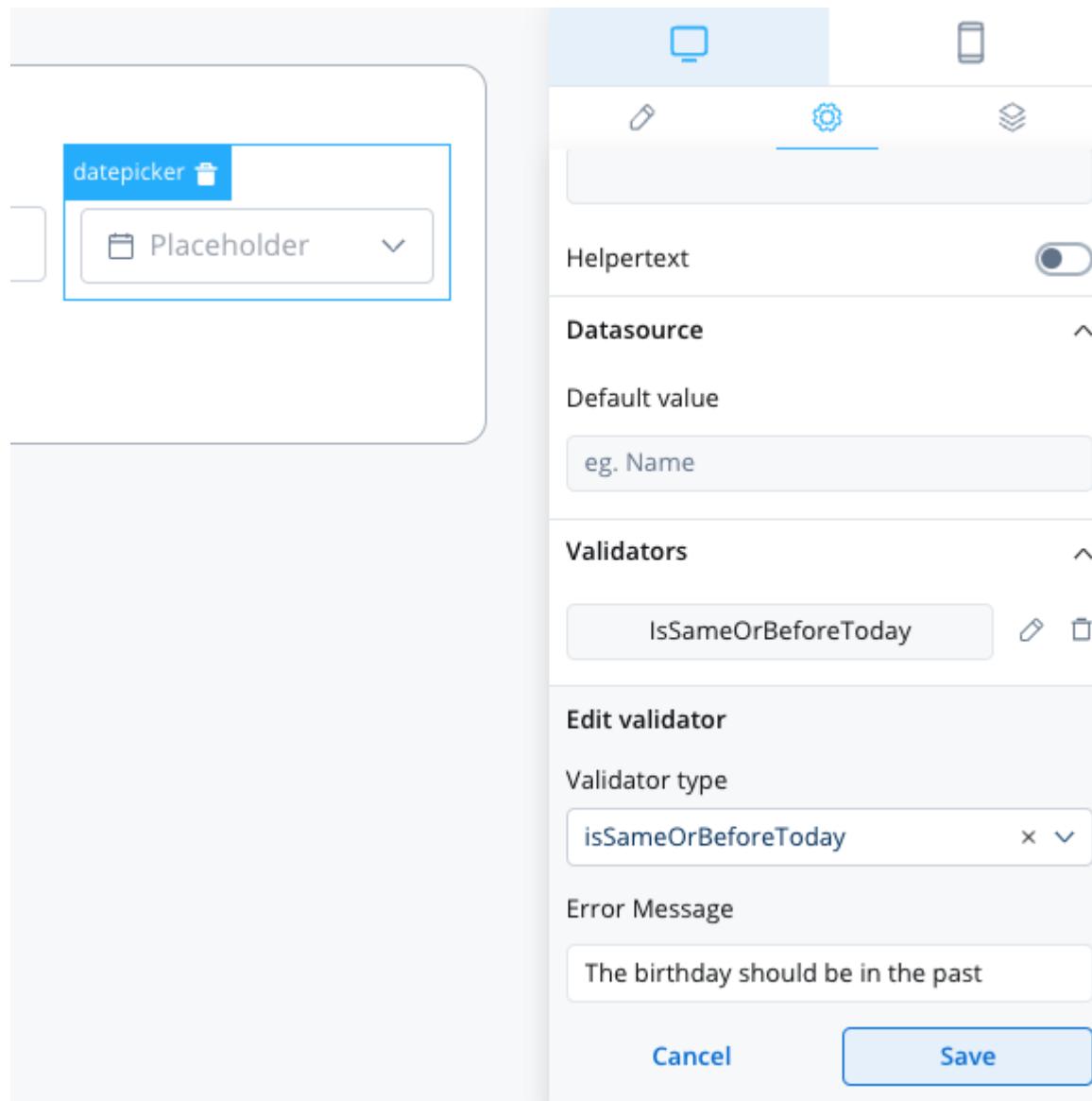
pattern validator

This validator checks whether the input value matches the specified pattern (for example, a [regex expression](#)).



datepicker - isSameOrBeforeToday

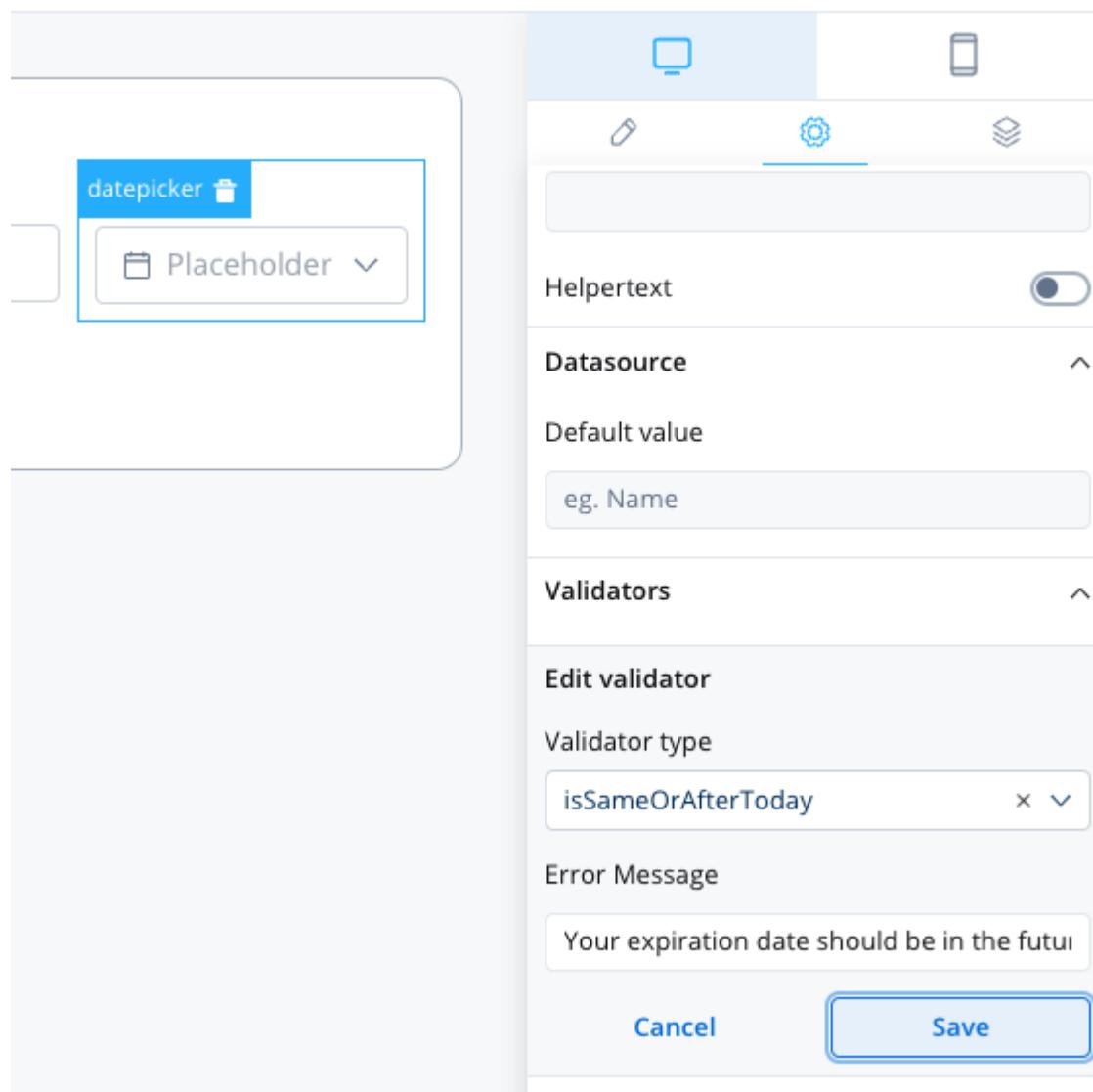
This validator can be used to validate **datepicker** inputs. It checks whether the selected date is today or in the past. If there are no characters at all, this validator will not trigger. It is advisable to use this validator with a **required** validator.



datepicker - isSameOrAfterToday

This validator can be used to validate datepicker inputs. It checks whether the selected date is today or in the future. If there are no characters at all, this

validator will not trigger. It is advisable to use this validator with a **required** validator.



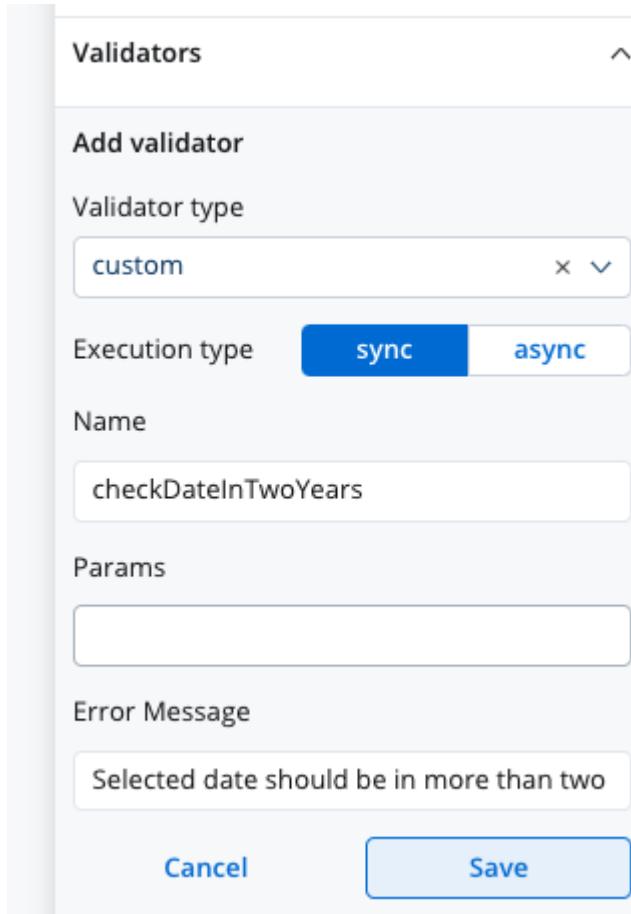
Custom validators

Additionally, custom validators can be created within the web application and referenced by name. These custom validators can have various configurations such as execution type, name, parameters, and error message.

1. **Execution type** - sync/async validator (for more details check [this](#))
2. **Name** - name provided by the developer to uniquely identify the validator
3. **Params** - if the validator needs inputs to decide if the field is valid or not, you can pass them using this list
4. **Error Message** - the message that will be displayed if the field is not valid

INFO

The error that the validator returns **MUST** match the validator name.



A screenshot of a web-based configuration interface for a custom validator. The interface has a light blue header bar with the title "Validators". Below the header, there's a section titled "Add validator" with a "Validator type" dropdown set to "custom". Under "Execution type", the "sync" button is highlighted in blue, while "async" is in grey. The "Name" field contains the value "checkDateInTwoYears". The "Params" field is empty. The "Error Message" field contains the text "Selected date should be in more than two". At the bottom, there are "Cancel" and "Save" buttons, with "Save" being the active one.

Custom validator example

Below you can find an example of a custom validator (`currentOrLastYear`) that restricts data selection to the current or the previous year:

currentOrLastYear

```
currentOrLastYear: function currentOrLastYear(AC:  
AbstractControl): { [key: string]: any } {  
  if (!AC) {  
    return null;  
  }  
  
  const yearDate = moment(AC.value, YEAR_FORMAT, true);  
  const currentDateYear = moment(new  
Date()).startOf('year');  
  const lastYear = moment(new Date()).subtract(1,  
'year').startOf('year');  
  
  if (!yearDate.isSame(currentDateYear) &&  
!yearDate.isSame(lastYear)) {  
    return { currentOrLastYear: true };  
  }  
  
  return null;
```

smallerOrEqualsToNumber

Below is another custom validator example that returns `AsyncValidatorFn` param, which is a function that can be used to validate form input asynchronously. The validator is called `smallerOrEqualsToNumber` and takes an array of `params---

sidebar_position: 3

as an input.

ⓘ INFO

For this custom validator the execution type should be marked as `async` using the UI Designer.

```
export function smallerOrEqualsToNumber (params$:  
Observable<any>[]): AsyncValidatorFn {  
    return (AC): Promise<ValidationErrors | null> |  
Observable<ValidationErrors | null> => {  
    return new Observable((observer) => {  
        combineLatest(params$).subscribe(([maximumLoanAmount])  
=> {  
            const validationError =  
                maximumLoanAmount === undefined || !AC.value ||  
Number(AC.value) <= maximumLoanAmount ? null :  
{smallerOrEqualsToNumber: true};  
  
            observer.next(validationError);  
            observer.complete();  
        });  
    });  
};  
}
```

If the input value is undefined or the input value is smaller or equal to the maximum loan amount value, the function returns `null`, indicating that the input is valid. If the input value is greater than the maximum loan amount value, the

function returns a `ValidationErrors` object with a key `smallerOrEqualsToNumber` and a value of true, indicating that the input is invalid.

 **INFO**

For more details about custom validators please check this link.

Using validators in your application can help ensure that the data entered by users is valid, accurate, and consistent, improving the overall quality of your application.

It can also help prevent errors and bugs that may arise due to invalid data, saving time and effort in debugging and fixing issues.

Overall, enforcing data validation using validators is a crucial step in building high-quality, reliable, and user-friendly applications.

Was this page helpful?

BUILDING BLOCKS / UI Designer / Dynamic & computed values

In modern application development, the ability to create dynamic and interactive user interfaces is essential for delivering personalized and responsive experiences to users. Dynamic values and computed values are powerful features that enable developers to achieve this level of flexibility and interactivity.

Dynamic values

Dynamic values refer to the capability of dynamically populating element properties in the user interface based on process parameters or substitution tags. These values can be customized at runtime, allowing the application to adapt to specific scenarios or user input. With dynamic values, you can personalize labels, placeholders, error messages, and other properties of UI elements, providing a tailored experience for users.

You can now utilize process parameters or **substitution tags** with the following UI elements and their properties:

Element	Property	Accepts Params/Subst Tags
Form Elements	Default Value (except switch)	Yes
	Label, Placeholder	Yes
	Helper Text, Validators	Yes
Document Preview	Title, Subtitle	Yes
Card	Title, Subtitle	Yes
Form	Title	Yes
Message	Message	Yes

Element	Property	Accepts Params/Subst Tags
Buttons	Label	Yes
Select, Checkbox, Radio, Segmented Button (Static)	Label, Value	Subst Tags Only
Text	Text	Yes
Link	Link Text	Yes
Modal	Modal Dismiss Alert, Title, Message, Confirm Label, Cancel Label	Yes
Step	Label	Yes

Example using Substitution tags

The screenshot shows the FLOWX.AI builder interface. On the left, there's a sidebar with categories like Layout, Forms, and Collection, each containing various UI element icons. The main area displays a process step. The step contains an input field with the placeholder "This is a substitution tag". Below the input field, there are three fields: "prefix_en", "placeholder_en", and "suffix_en", each with their respective helpertexts. A success message "All data has been validated. Thank you!" is displayed below the input field. To the right of the process step is a configuration panel for the "Input" element. It includes tabs for Desktop and Mobile, and sections for Input, Properties, and Validators. The Input section shows "Process data key" set to "inputKey". The Properties section shows "Label" set to "@@docs_label", "Placeholder" set to "@@placeholder", "Type" set to "text", "Prefix" set to "@@prefix", and "Suffix" set to "@@suffix". The Validators section shows a "Default value" field with "f(x)" checked and "@@default_value" entered.

Example using process parameters

Business rule example

In the above example, the following MVEL business rule was used to populate the keys with values from the task:

```
//assigning a JSON object containing dynamic values for the
specified keys to the "app" key

output.put("app", {"label": "This is a label",
                  "title": "This is a title",
                  "placeholder": "This is a placeholder",
                  "helpertext": "This is a helper text",
```

```
"errorM":"This is a error message",
"prefix":"prx",
"suffix":"sfx",
"subtile":"This is a subtitle",
"message":"This is a message",
"defaultV":"defaultValue",
"value":"Value101",
"value":"Value101",
"confirmLabel":"This is a confirm
label",
"cancelLabel":"This is a cancel label",
"defaultValue":"dfs",
"defaultDate":"02.02.2025",
"defaultSlider": 90});
```

⚠ CAUTION

Please note that for releases **<= 3.3.0**, it is not possible to concatenate process parameters with substitution tags when using dynamic values.

Computed values

Computed values take the concept of dynamic values a step further by allowing you to generate values dynamically using JavaScript expressions. Rather than relying solely on predefined values, computed values enable the calculation, transformation, and manipulation of data based on specific rules or conditions.

The screenshot shows the FLOWX.AI application interface. On the left is a sidebar with various building blocks categorized under 'Layout', 'Forms', and 'Collection'. In the center, there's a form titled 'Enter Personal Information' with fields for First Name, Last Name, Date of birth, Employment type (radio buttons for Employed and Pensioner), Income range (a slider from < \$50.000 to \$50.000 - \$100.000), and a Save personal information toggle. Below these are fields for Loan amount (text input) and Down payment (a slider with values NaN \$). A modal window titled 'Computed expression' is open over the form, containing a JavaScript code block:

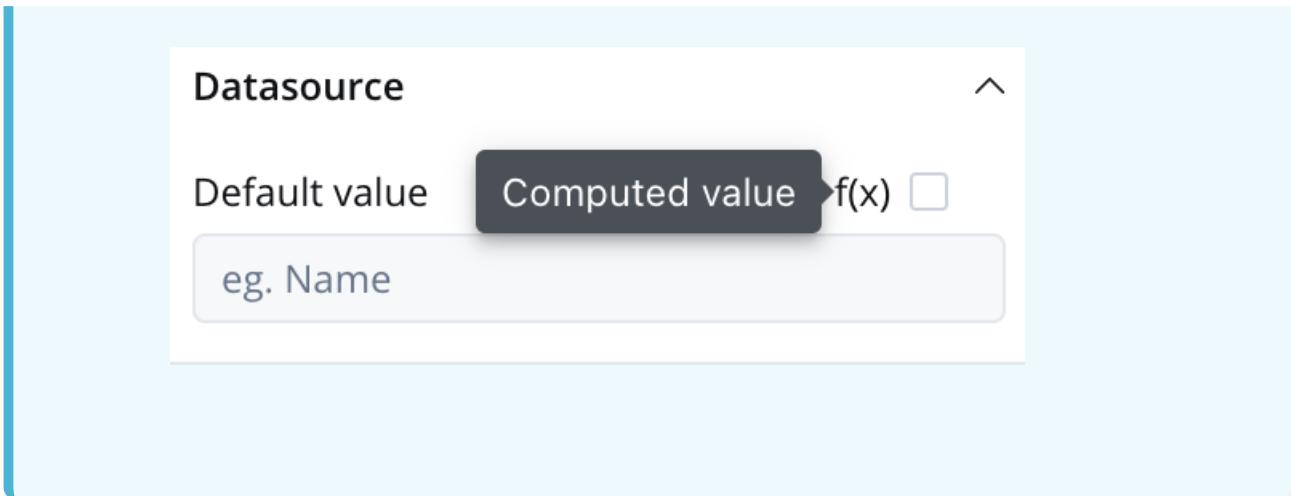
```
1 if ( !isNaN(parseInt(${application.client.amount})) ) {
2     return 0.15 * parseInt(${application.client.amount})
3 } else {
4     return 2000
5 }
6
```

The modal also has 'Cancel' and 'Save' buttons, and sections for 'Default value' (with placeholder 'eg. Name') and 'Validators' (with a 'Add a validator' button).

Computed values can be created by writing JavaScript expressions that operate on process parameters or other variables within the application.

INFO

To add a computed value, you have to explicitly check “Computed value” option (represented by the **f(x)** icon), which will transform the desired field into a JavaScript editor.



By enabling computed values, the application provides flexibility and the ability to create dynamic and responsive user interfaces.

Slider Configuration:

```

Slider
Process data key
application.client.avans

General properties
Slider label
Down payment
Show value label

Helpertext
Min Value f(x) ✎
if (
!isNaN(parseInt($("#application.client.amount")))
) {
return 0.15 *
parseInt($("#application.client.amount"))
} else {
}

Max Value f(x) ✎
if (
!isNaN(parseInt($("#application.client.amount")))
) {
return 0.35 *
parseInt($("#application.client.amount"))
} else {
}

Suffix
$ Step size
1000

Datasource
Default value f(x) ✎
eg. Name

```

Slider example

The above example demonstrates the usage of computed values for a Slider element, where JavaScript expressions are used to compute the minimum and maximum values based on a value entered in an input UI element (linked by the process key `${application.client.amount}`).

Min Value

```
if ( !isNaN(parseInt(${application.client.amount})) ) {  
    return 0.15 * parseInt(${application.client.amount})  
} else {  
    return 10000  
}
```

Max Value

```
if ( !isNaN(parseInt(${application.client.amount})) ) {  
    return 0.35 * parseInt(${application.client.amount})  
} else {  
    return 20000  
}
```

Example details

The code snippets check whether the value of

`${application.client.amount}` key can be successfully parsed as an integer. Here's a step-by-step explanation:

- The `parseInt()` function is used to attempt to convert `${application.client.amount}` into an integer.

- The `isNaN()` function is then used to check if the result of the conversion is `NaN` (not a number).
- If the value is not `NaN`, it means `${application.client.amount}` is a valid numeric value.
- In that case, the code calculates the computed value by multiplying the parsed integer by `0.15` (the minimum percentage value for the down payment or with 0.35, the maximum percentage value of the down payment). The result is returned as the computed value for the expression.
- If the value is `NaN` (or `${application.client.amount}` couldn't be successfully parsed as an integer), the code executes the `else` block and returns a default value of `10000`.

In summary, the JS expressions demonstrates how a computed value can be derived based on a conditional calculation. It first checks if a specific process parameter (`${application.client.amount}`) is a valid numeric value, and if so, it computes the value by multiplying it by 0.15 or 0.35. Otherwise, it falls back to a default value of `10000` or `20000`.

Usage

The UI Designer now allows JavaScript expressions to create computed values used on the following UI elements with their properties:

Element	Properties
Slider	min Value, max Value, default Value
Input	Default Value

Element	Properties
Any UI Element that accepts validators	min, max, minLength, maxLength
Text	Text
Link	Link Text

- **Slider:** The min value, max value, and default value for sliders can be set using JavaScript expressions applied to process parameters. This allows for dynamic configuration based on numeric values.
- **Any UI Element that accepts validators min, max, minLength, maxLength:** The "params" field for these elements can also accept JavaScript expressions applied to process parameters. This enables flexibility in setting validator parameters dynamically.
- **Default Value:** For input elements like text inputs or number inputs, the default value can be a variable from the process or a computed value determined by JavaScript expressions.
- **Text:** The content of a text element can be set using JavaScript expressions, allowing for dynamic text generation or displaying process-related information.
- **Link:** The link text can also accept JavaScript expressions, enabling dynamic generation of the link text based on process parameters or other conditions.

Please note that these settings are specifically applicable to numeric values and are not intended for date or string values.

CAUTION

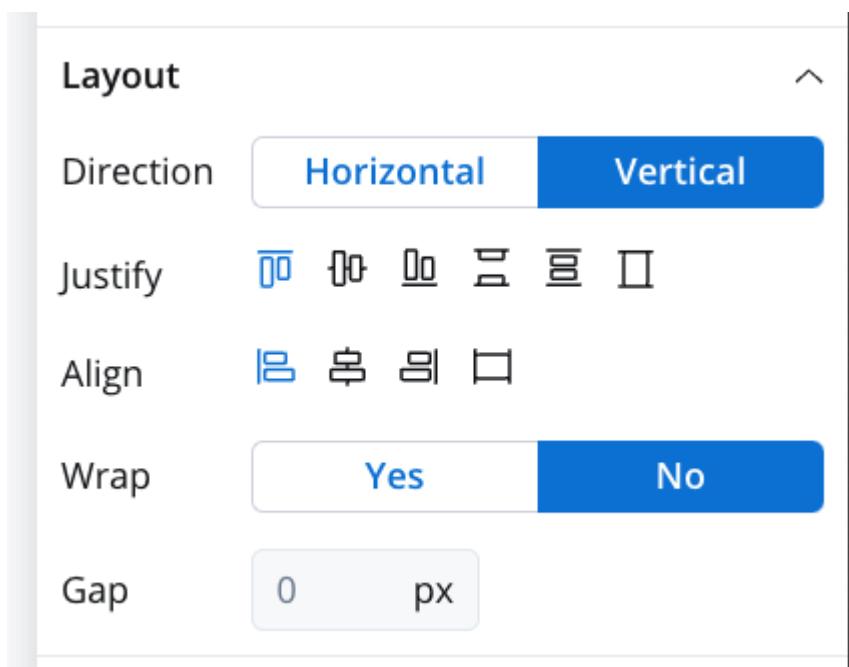
For input elements (e.g., text input), you may require a default value from a process variable, while a number input may need a computed value.

Was this page helpful?

BUILDING BLOCKS / UI Designer / Layout configuration

Layout settings will be available for all components that can group other types of elements (for example, [Container](#) or [Card](#)).

These settings allow users to customize properties as the layout direction, alignment, gap, sizing, and spacing.



These settings can be applied practically in various ways, depending on the context and purpose of the design. For example:

- The layout direction and alignment settings can be used to ensure that the content is displayed in a logical and visually appealing way. For instance, a left-to-right layout direction may be more appropriate for languages that read from left to right, while a center alignment may be better for headings or titles.
- The gap, sizing, and spacing settings can be used to control the distance between elements in a design. This can help create a sense of hierarchy and balance, as well as improve readability and usability. For example, increasing the spacing between paragraphs or sections can make the content easier to scan and read, while reducing the size of certain elements can help prioritize others.
- Customizing these properties can also help ensure that a design is accessible and inclusive. For instance, adjusting the layout direction and alignment settings can make the design more readable for users with certain disabilities or who use assistive technologies. Similarly, increasing the spacing and sizing of interactive elements can make them easier to click or tap for users with motor impairments.

To better understand how these layout configurations work and see real-time feedback on different combinations, please refer to the following link:

» [Layout configuration](#)

Was this page helpful?

BUILDING BLOCKS / UI Designer / Rendering and UI Designer changelog

⚠ CAUTION

This changelog is relevant to the 3.0 release, which introduces significant changes to the UI configuration.

Please be aware that a process (when it comes to UI configuration) may look different from previous releases and that certain updates may not be compatible with older configurations. The migration process may take longer than usual.

Notes for post-migration

After migrating to the 3.0 release, please take in consideration the following changes to ensure smooth operation:

1. Verify font sizes where float values were set.
2. Verify line heights where scale values were set.
3. Verify border radius values where values other than px were set.
4. Review and set padding and margin values where needed. Deleted keys include "margin" : "8px 0" and "padding" : "16px 0 0 16px".
5. Check **radio** and **checkbox** elements and update the new `label` prop that has been added.
6. Configure the new `column` prop for Layout (available for checkbox and radio), which allows for grouping many enumerations.
7. The `height` prop (from **Container**, **Form** and **Card**) has been removed.

8. Update the width prop by configuring the new `Fit W` prop with values such as fill, fixed, or auto. `Min H` and `Max H` props have been removed.
9. The hint and message UI components were combined into a single component called 'message' with the following properties: `type`, `fill`, `border`, and `text`.
10. The `button` element no longer has the `fill`, `border`, and `flat` types. It now has 4 types: `primary`, `secondary`, `ghost`, and `text`.
11. Added Helpertext (to replace Info tooltip) - this new element can be found on [Form elements](#) and provides additional information about each element, which can be hidden within an infopoint.

[Was this page helpful?](#)

BUILDING BLOCKS / Token

Token is the concept that describes the current position in the process flow. When you start the process you have a graph of [nodes](#) and based on the configuration you will go from one to another based on the defined sequence (connection between nodes).

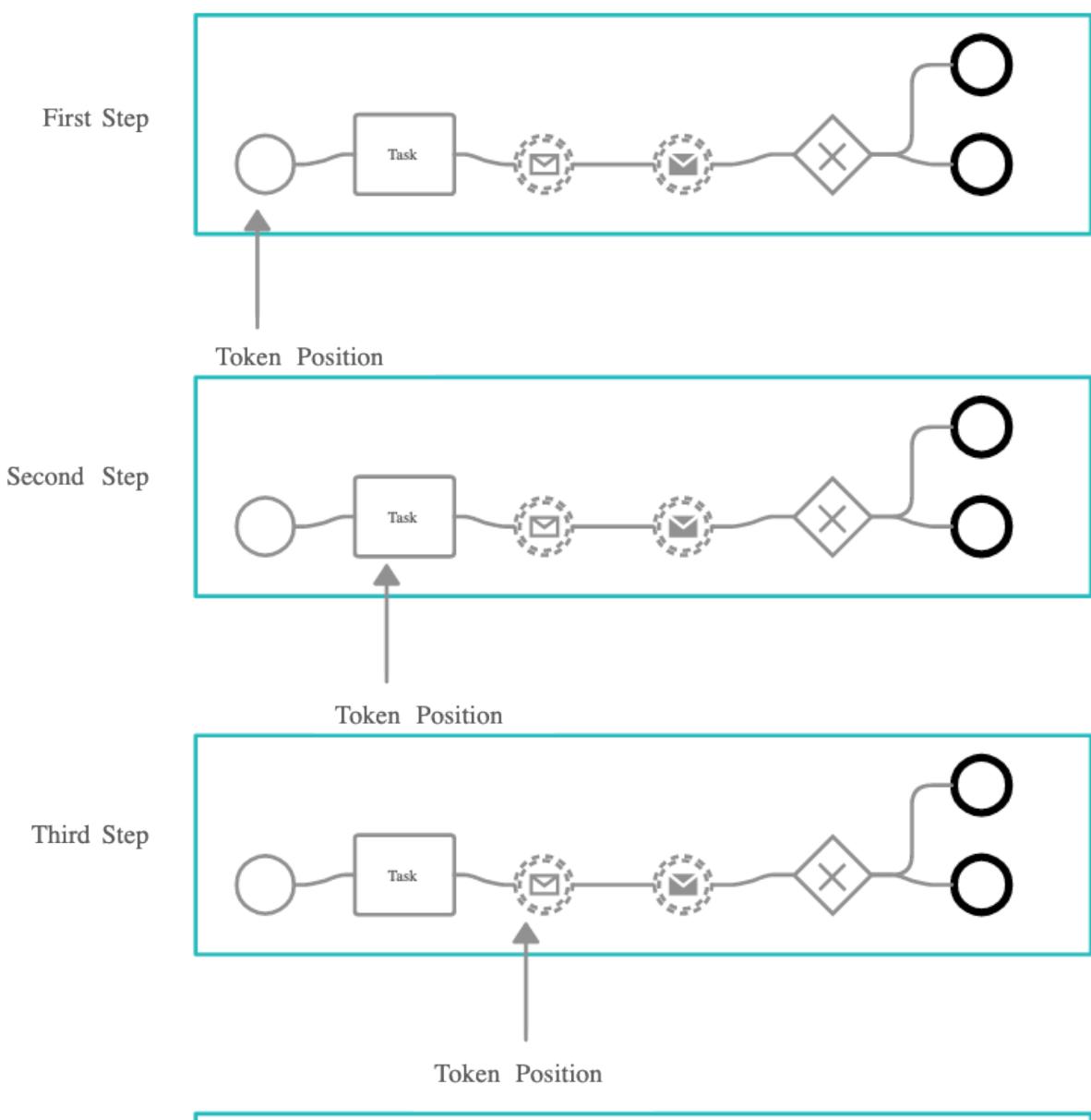
The token is a [BPMN](#) concept that represents a state within a process instance. It keeps track of the current position in the process flow and is used to store data related to the current process instance state.

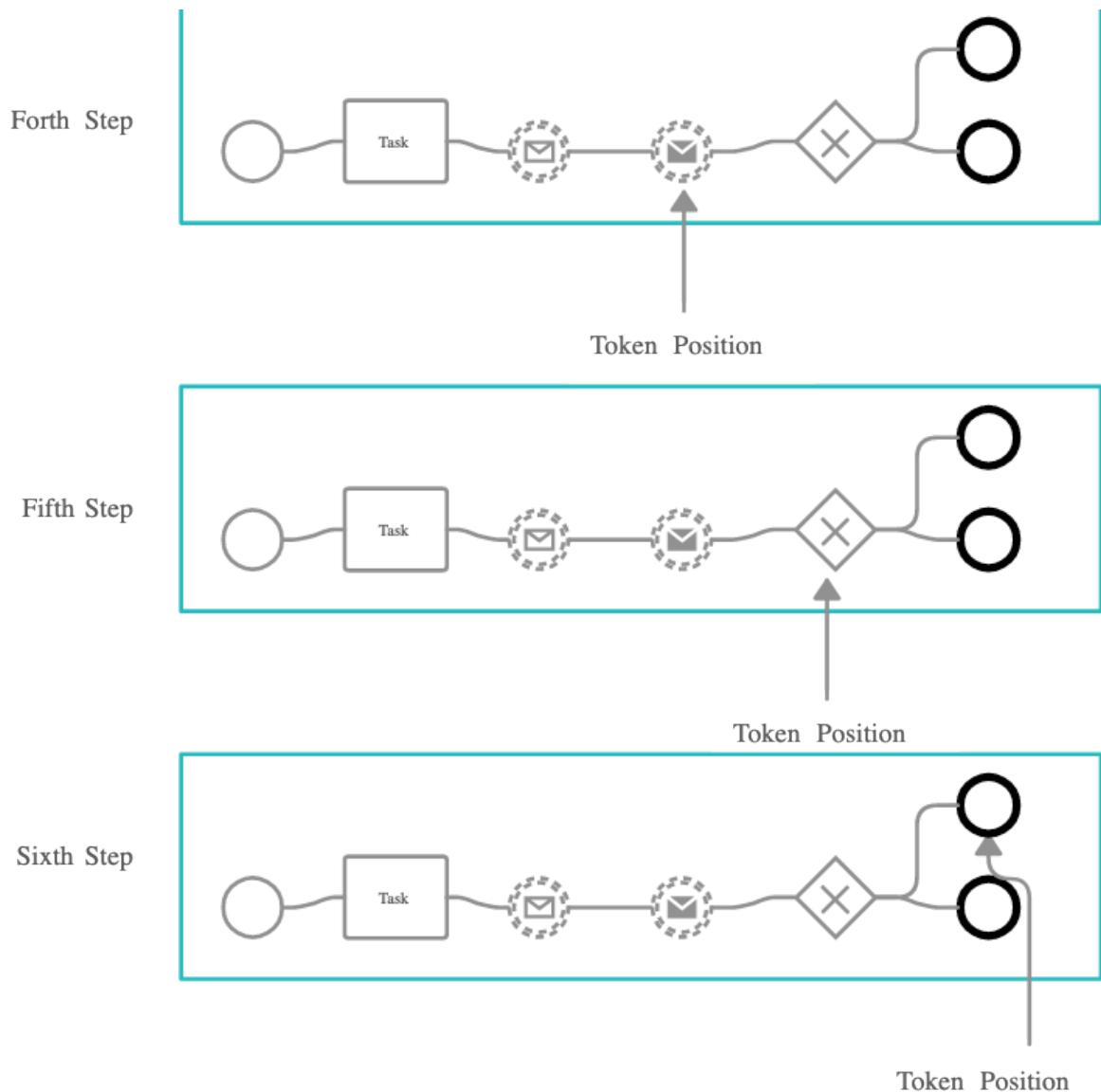
A token is created each time a new process instance is started. As the actions on the process instance are executed, the token advances from one node to the next.

As a node can have several **actions** that need to be executed, the token is also used for keeping track of the actions executed in each node.

In case of **parallel gateways**, child tokens are created for each flow branch. The parent token moves to the gateway sync node and only advances after all the child tokens also reach that node.

The image below shows how a token advances through a process flow:





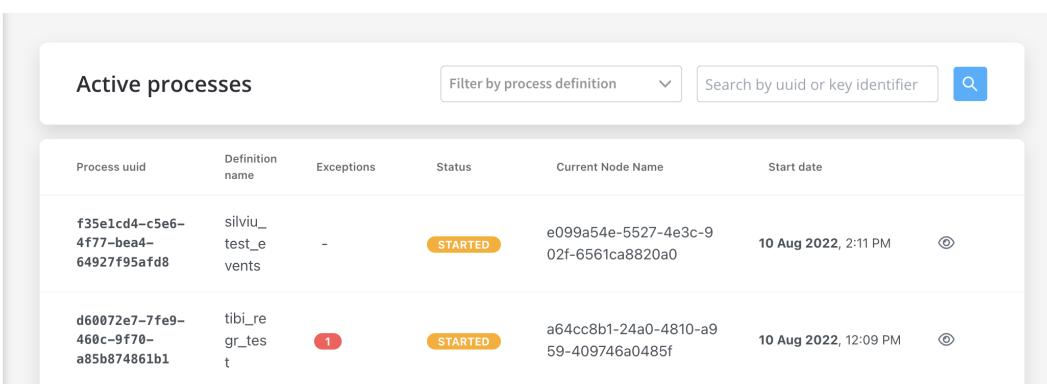
The token will only move to the next node when there are no more mandatory actions from the current node that need to be executed. The token will also wait on a node in case the node is set to receive an event from an external system through Kafka.

There will be cases when the token needs to be stopped in a node until some input is received from the user. If the input from the user is needed for further

advancing in the process, the token should only advance after all data was received. A mandatory manual action can be used in this case and linked to the user action. This way we make sure that the process flow advances only after the user input is received.

Checking the token status

The current process instance status can be retrieved using the FLOWX Designer. It will display some info on the tokens related to that process instance and the current nodes they are in.



The screenshot shows the FLOWX Designer interface with the following details:

- Active processes:** A table listing two active process instances.
- Columns:** Process uuid, Definition name, Exceptions, Status, Current Node Name, and Start date.
- Process 1:** Process uuid: f35e1cd4-c5e6-4f77-bea4-64927f95afd8, Definition name: silviu_test_events, Status: STARTED, Current Node Name: e099a54e-5527-4e3c-902f-6561ca8820a0, Start date: 10 Aug 2022, 2:11 PM.
- Process 2:** Process uuid: d60072e7-7fe9-460c-9f70-a85b874861b1, Definition name: tibi_regr_test, Status: STARTED, Current Node Name: a64cc8b1-24a0-4810-a959-409746a0485f, Start date: 10 Aug 2022, 12:09 PM.

In case more details are needed about the token, you can click the **Process status** view button, choose a token then click the **view button** again:

The screenshot shows the FLOWX.AI interface. On the left, there's a sidebar with 'Processes' (Definitions, Active process, Process Instances, Failed process start) and 'Content Management' (Enumerations, Substitution tags, Content models, Languages). A user profile for 'Dragos Caravasile' is at the bottom. The main area is titled 'Process Definitions' with a search bar. It has two sections: 'Drafts / In progress' and 'Published'. The 'Drafts / In progress' section contains two entries:

Name	Version	Edited at	Edited by
AutoTestProcess72860032	1	29 Jun 2022, 1:49 PM	QA FlowX
TA_BackinSteps_Process_1654862288218	10	29 Jun 2022, 11:33 AM	QA FlowX

The 'Published' section contains two entries:

Name	Version	Published at	Published by
TA_Subprocess_2	4	29 Jun 2022, 11:40 AM	QA FlowX
TA_Subprocess_item_2	4	29 Jun 2022, 11:40 AM	QA FlowX

Token status details

The following token details are available when you access and view the JSON file of a token in FLOWX Designer:

```
id: 492952
version: 31
parentTokenId: null
currentNodeId: 491660
currentnodeName: null
state: "INACTIVE"
statusCurrentNode: "EXECUTED_COMPLETE"
syncNodeTokensCount: 0
syncNodeTokensFinished: 0
dateUpdated: "2022-05-18T09:57:58.639911Z"
paramValues: null
processInstanceId: 492902
currentNode: null
nodesActionStates:
  0: Object {"nodeId":491663,"name":"Start","arrivedDate":"2022-05-18T09:57:58.639911Z","lastTransitionDate":"2022-05-18T09:57:58.639911Z","status":1,"action":{},"value":{},"error":{},"script":{},"scriptResult":{}}
```

```
1: Object {"nodeId":491657,"name":"stepper","arrivedDate":"2023-07-26T09:57:58.085Z","lastExecutionDate":null,"state":"COMPLETED","lastExecutionTime":1684980278000,"lastExecutor":491664,"actionStateData":null}
2: Object {"nodeId":491656,"name":"step1","arrivedDate":"2023-07-26T09:57:58.085Z","lastExecutionDate":null,"state":"COMPLETED","lastExecutionTime":1684980278000,"lastExecutor":491664,"actionStateData":null}
3: Object {"nodeId":491662,"name":"Client Form","arrivedDate":null,"lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
{"492053": {"name": "saveClient", "state": "COMPLETED", "lastExecutionDate": null, "lastExecutionTime": 1684980278000, "lastExecutor": 491664, "actionStateData": null}}
4: Object {"nodeId":491664,"name":"end step1","arrivedDate":"2023-07-26T09:57:58.085Z","lastExecutionDate":null,"state":"COMPLETED","lastExecutionTime":1684980278000,"lastExecutor":491664,"actionStateData":null}
5: Object {"nodeId":491661,"name":"step2","arrivedDate":"2023-07-26T09:57:58.085Z","lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
6: Object {"nodeId":491655,"name":"company form","arrivedDate":null,"lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
{"492052": {"name": "SaveCompany", "state": "COMPLETED", "lastExecutionDate": null, "lastExecutionTime": 1684980278000, "lastExecutor": 491664, "actionStateData": null}}
7: Object {"nodeId":491658,"name":"stop step2","arrivedDate":null,"lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
8: Object {"nodeId":491659,"name":"stop_stepper","arrivedDate":null,"lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
9: Object {"nodeId":492452,"name":"CreateDocument","arrivedDate":null,"lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
{"492102": {"name": "sendInformation", "state": "COMPLETED", "lastExecutionDate": null, "lastExecutionTime": 1684980278000, "lastExecutor": 491664, "actionStateData": null}}
10: Object {"nodeId":492453,"name":"ReceiveDocuments","arrivedDate":null,"lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
18T09:57:58.085Z", "actionStateData": null}
11: Object {"nodeId":491660,"name":"end_process","arrivedDate":null,"lastExecutionDate":null,"state":null,"lastExecutionTime":0,"lastExecutor":null,"actionStateData":null}
backSeq: Object {"nodes": [491663, 491657, 491656, 491662, 491664, 491661, 491658, 491659, 492452, 492102, 492453, 491660]}
uuid: "794954a7-875f-4508-bbcb-8a11cf7a9b37"
```

Token status details		Execution details
id	492952	2023-07-26T09:57:58.085Z
version	31	2023-07-26T09:57:58.085Z
parentTokenId	null	2023-07-26T09:57:58.085Z

Token status details		Ex...
currentNodeId	491660	
state	ACTIVE, ON_HOLD, INACTIVE	
statusCurrentNode	ARRIVED, EXECUTING, EXECUTED_PARTIAL, EXECUTED, MESSAGE_RESPONSE_TIMED_OUT	
syncNodeTokensCount	syncNodeTokensCount: 0	

Token status details	
syncNodeTokensFinished	syncNodeTokensFinished: 0
dateUpdated	"2022-05-18T09:53:28.587930Z"

Token status details	Ex...
processInstanceId	492902
nodesActionStates	<pre>0: Object {"nodeId":491663,"name":"Start", "actionState": "PENDING", "lastModified": "2023-07-18T09:56:39.576Z", "actionStateData": null}</pre>
backSeq	<pre>Object {"nodes": [491663, 491657, 491656, 491662, 491664, 491665]}</pre>

Was this page helpful?

BUILDING BLOCKS / Supported scripts

Supported scripts

Scripts are used to define and run **actions** but also properties inside **nodes**. For now, the following script languages are supported:

- Python (Jython)
- DMN
- MVEL
- Groovy
- JavaScript (Nashorn Engine)

Scripting Language	Version
Python (Jython)	2.7.0
DMN	1.3
MVEL	2.4.10
Groovy	3.0.8

Scripting Language	Version
Nashorn engine (JavaScript)	15.4

Python

(!) INFO

We use **Jython**.

Jython is an implementation of the high-level, dynamic, object-oriented language **Python** seamlessly integrated with the **Java** platform. Jython is an open-source solution.

Properties:

- Supports **Python 2.7** most common python libs can be imported, ex: math, time, etc.
- Java libs can also be imported: [details here](#)

Useful links:

» [Python 2.7.18 documentation](#)

» [Jython](#)

» [Jython FAQs](#)

DMN

Decision Model and Notation (DMN) is a standard for Business Decision Management.

FLOWX uses [BPMN.io](#) (based on **camunda-engine-dmn** version **7.14.0**) which is built on [DMN 1.3](#) standards.

Properties:

camunda-engine-dmn supports [DMN 1.3](#), including Decision Tables, Decision Literal Expressions, Decision Requirements Graphs, and the Friendly Enough Expression Language (FEEL)

Useful links:

» [Decision Model and Notation \(DMN\)](#)

» [DMN 1.3 specs](#)

More information:

» [Intro to DMN](#)

» DMN Business Rule Action

MVEL

MVEL is a powerful expression language for Java-based applications. It provides a plethora of features and is suited for everything from the smallest property binding and extraction, to full-blown scripts.

- FLOWX uses **mvel2 - 2.4.10 version**

Useful links:

» Mvel documentation

» Maven repository: Mvel 2.4.0 final

More information:

» Intro to MVEL

Groovy

Groovy is a multi-faceted language for the Java platform. The language can be used to combine Java modules, extend existing Java applications and write new

applications

We use and recommend **Groovy 3.0.8** version, using **groovy-jsr223** engine.

(!) INFO

Groovy has multiple ways of integrating with Java, some of which provide richer options than available with **JSR-223** (e.g. greater configurability and more security control). **JSR-223** is recommended when you need to keep the choice of language used flexible and you don't require integration mechanisms not supported by **JSR-223**.

(!) INFO

JSR-223 (spec) is a **standard scripting API for Java Virtual Machine (JVM) languages**. The JVM languages provide varying levels of support for the JSR-223 API and interoperability with the Java runtime.

Useful links:

» [Groovy Language Documentation](#)

» [\[Java\] Class GroovyScriptEngineImpl](#)

Nashorn Engine (JavaScript)

Nashorn engine is an open source implementation of the [ECMAScript Edition 5.1 Language Specification](#). It also implements many new features introduced in ECMAScript 6 including template strings; `let`, `const`, and block scope; iterators and `for..of` loops; `Map`, `Set`, `WeakMap`, and `WeakSet` data types; symbols; and binary and octal literals. It is written in Java and runs on the Java Virtual Machine.

Latest version of **Nashorn** is **15.4**, available from [Maven Central](#). You can check the [changelog](#) to see what's new.

Useful links:

» [GitHub - Nashorn](#)

» [OpenJDK - Nashorn](#)

Was this page helpful?