FLOWX.AI

BUILDING BLOCKS / Node / user-task-node

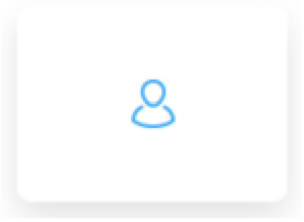
Contents

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.

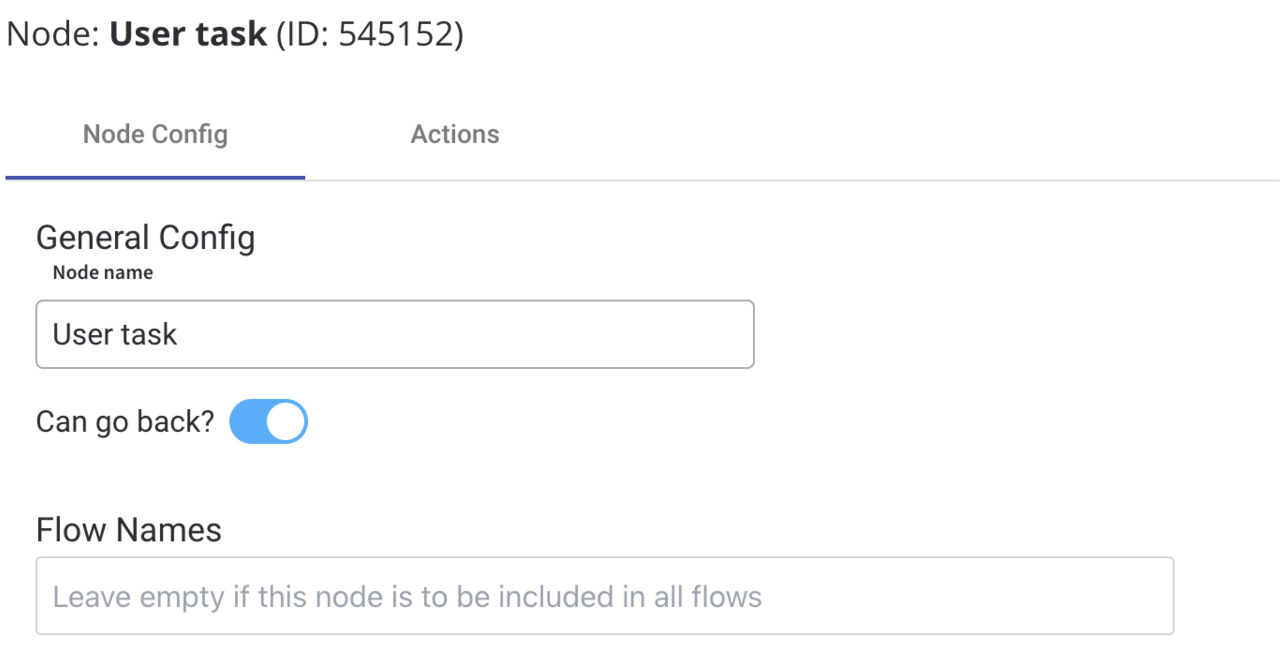
Configuring a user task node



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

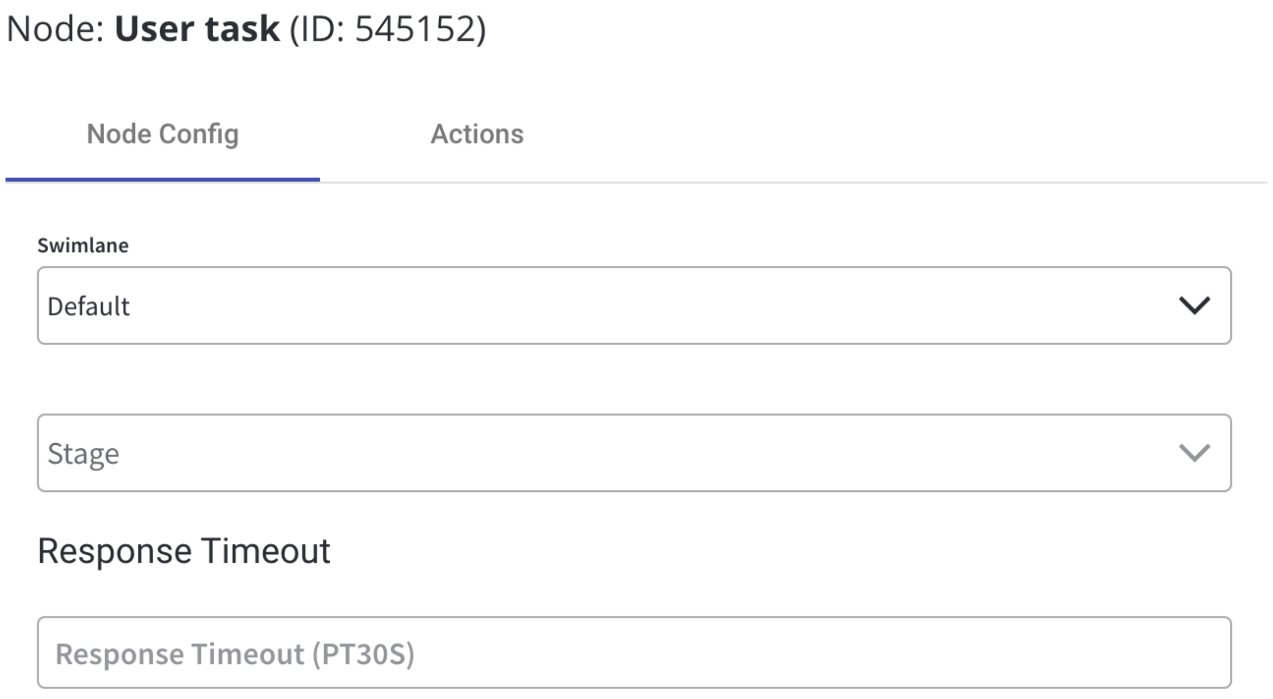
INFO

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)

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)

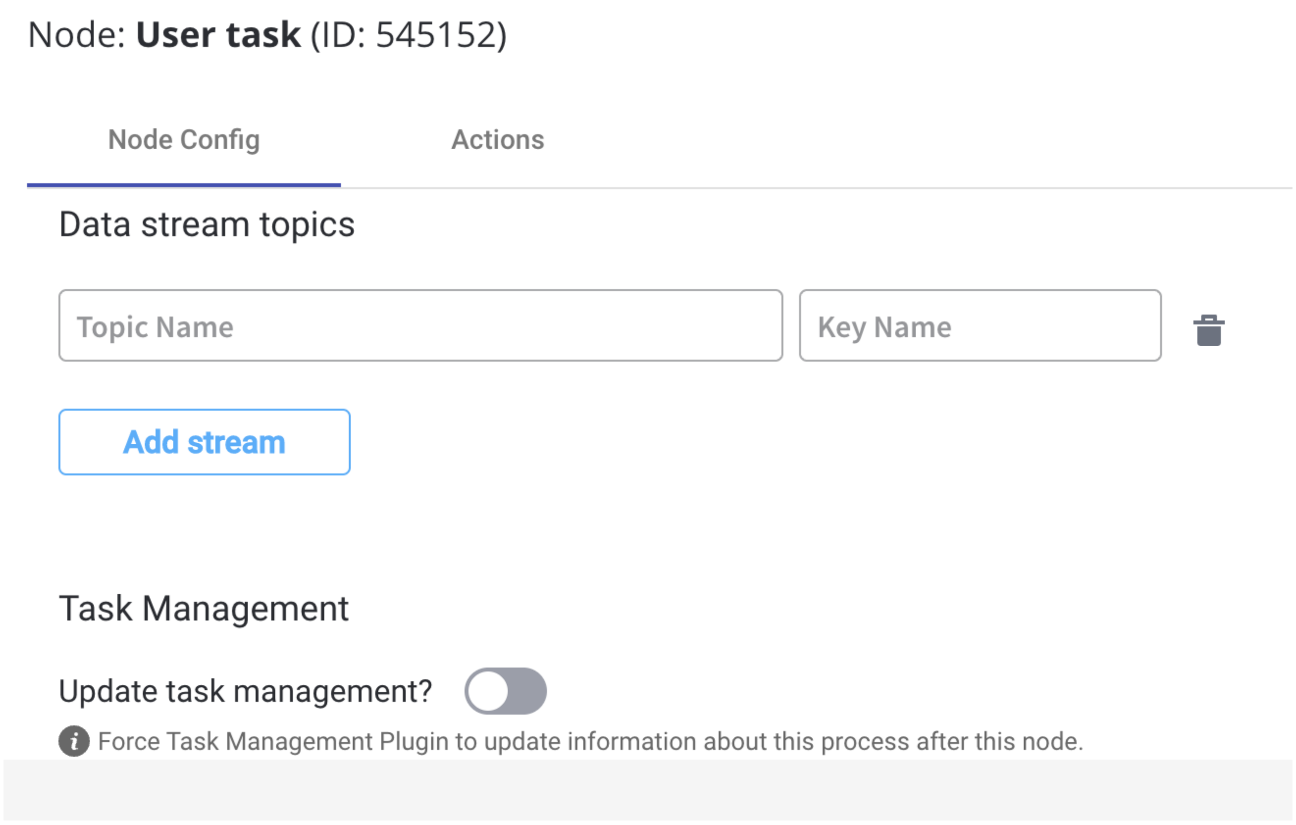
DANGER

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

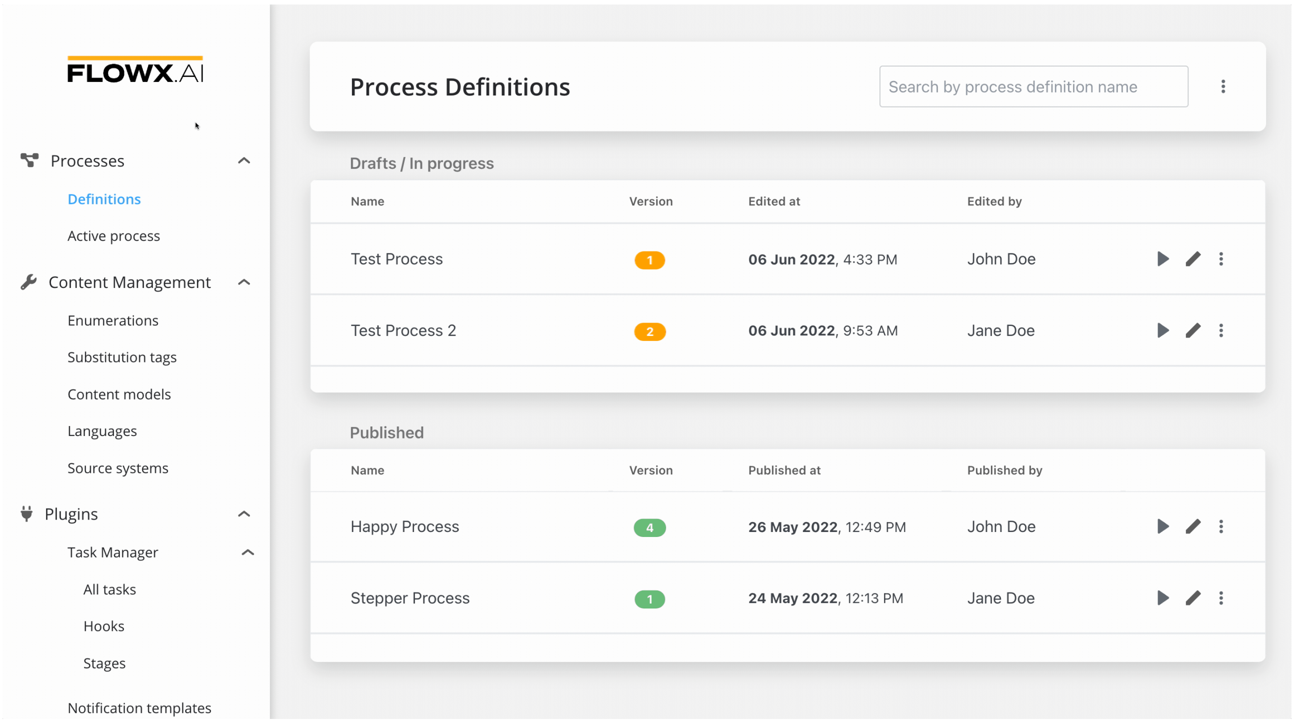
The fallback content to display on prerendering
includes an intuitive UI Designer (drag-and-drop editor) for creating diverse UI templates. You can use various elements from basic buttons, indicators, and forms, but also predefined collections or prototypes.

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**.



Process Definitions

Drafts / In progress

Name	Version	Edited at	Edited by
Test Process	1	06 Jun 2022, 4:33 PM	John Doe
Test Process 2	2	06 Jun 2022, 9:53 AM	Jane Doe

Published

Name	Version	Published at	Published by
Happy Process	4	26 May 2022, 12:49 PM	John Doe
Stepper Process	1	24 May 2022, 12:13 PM	Jane Doe

» Creating a user interface

Predefined components

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

Predefined components can be split in 3 categories:

1. Root components

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

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

More details in the following section:

» Root components

2. UI Components

The root component can hold a hierarchical component structure.

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

- **Container** - used to group and align its children

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

More details in the following section:

» Component types

3. Form elements

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

1. **Input** - FLOWX form element that allows you to generate an input form filed
2. **Select** - to add a dropdown
3. **Checkbox** - the user can select zero or more input from a set of options
4. **Radio** - the user is required to select one and only one input from a set of options

5. **Datepicker** - to select a date from a calendar picker
6. **Switch** - allows the user to toggle an option on or off

More details in the following section:

» Form elements

Custom components

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

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

More details in the following section:

» Custom

The sections that can be configured are as follows:

1. **Message** - configure what data will be pushed to the frontend application
2. **Input keys** - used to define the process model paths from which the components will receive its data
3. **UI Actions** - actions defined here will be made available to the custom component. Multiple actions can be configured on a custom component and

mapped to different triggers when developing it. Naming each action suggestively is important so the frontend engineer developing the component knows what actions should be triggered by certain events.

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

Displaying a UI element

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

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

Example:

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

INFO

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

```
{
"processDefinitionName" : "DemoProcess",
    "tokens" : [ {
      "id" : 759224,
      "startNodeId" : null,
      "currentNodeId" : 662807,
      "currentNodeName" : null,
      "state" : "ACTIVE",
      "statusCurrentNode" : "ARRIVED",
      "dateUpdated" : "2023-05-31T09:44:39.969634Z",
      "uuid" : "d310996d-f3b9-44e5-983d-3631c844409e"
    } ],
    "state" : "STARTED",
    "templateConfig" : [ {
      "id" : 630831,
      "flowxUuid" : "80ea0a85-2b0b-442a-a123-2480c7aa2dce",
      "nodeDefinitionId" : 662856,
      "componentIdentifier" : "CONTAINER",
      "type" : "FLOWX",
      "order" : 1,
      "canGoBack" : true,
      "displayOptions" : {
        "flowxProps" : { },
        "style" : null,
        "flexLayout" : {
          "fxLayoutGap" : 0,
          "fxLayoutAlign" : "start stretch",
          "fxLayout" : "column"
        },
        "className" : null,
        "platform" : "DEFAULT"
      },
      "templateConfig" : [ {
        "id" : 630832,
        "flowxUuid" : "38e2c164-f8cd-4f6e-93c8-39b7cdd734cf",
        "nodeDefinitionId" : 662856,
        "uiTemplateParentId" : 630831,
        "componentIdentifier" : "TEXT",
        "type" : "FLOWX",
        "order" : 0,
        "key" : "",
        "canGoBack" : true,
        "displayOptions" : {
          "flowxProps" : {
            "text" : "Demo text"
          },
          "style" : null,
          "flexLayout" : null,
          "className" : null,
          "platform" : "DEFAULT"
        },
        "expressions" : {
          "hide" : ""
        },
        "templateConfig" : [ ],
        "dataSource" : {
          "processData" : {
            "parentFlowxUuid" : null
          },
          "nomenclator" : {
            "parentFlowxUuid" : null
          }
        }
      } ]
    } ],
    "uuid" : "44177340-5ac6-4591-89ad-04df0815fbd0",
    "generalData" : null,
    "backCounter" : 0,
    "startedByActionId" : null,
    "subProcesses" : null,
    "subprocessesUuids" : null,
    "baseUrl" : null
  }
```

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

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

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

Values

For more details, please check the following page:

» Message send receive task

Was this page helpful?