FLOWX.AI

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins

Contents

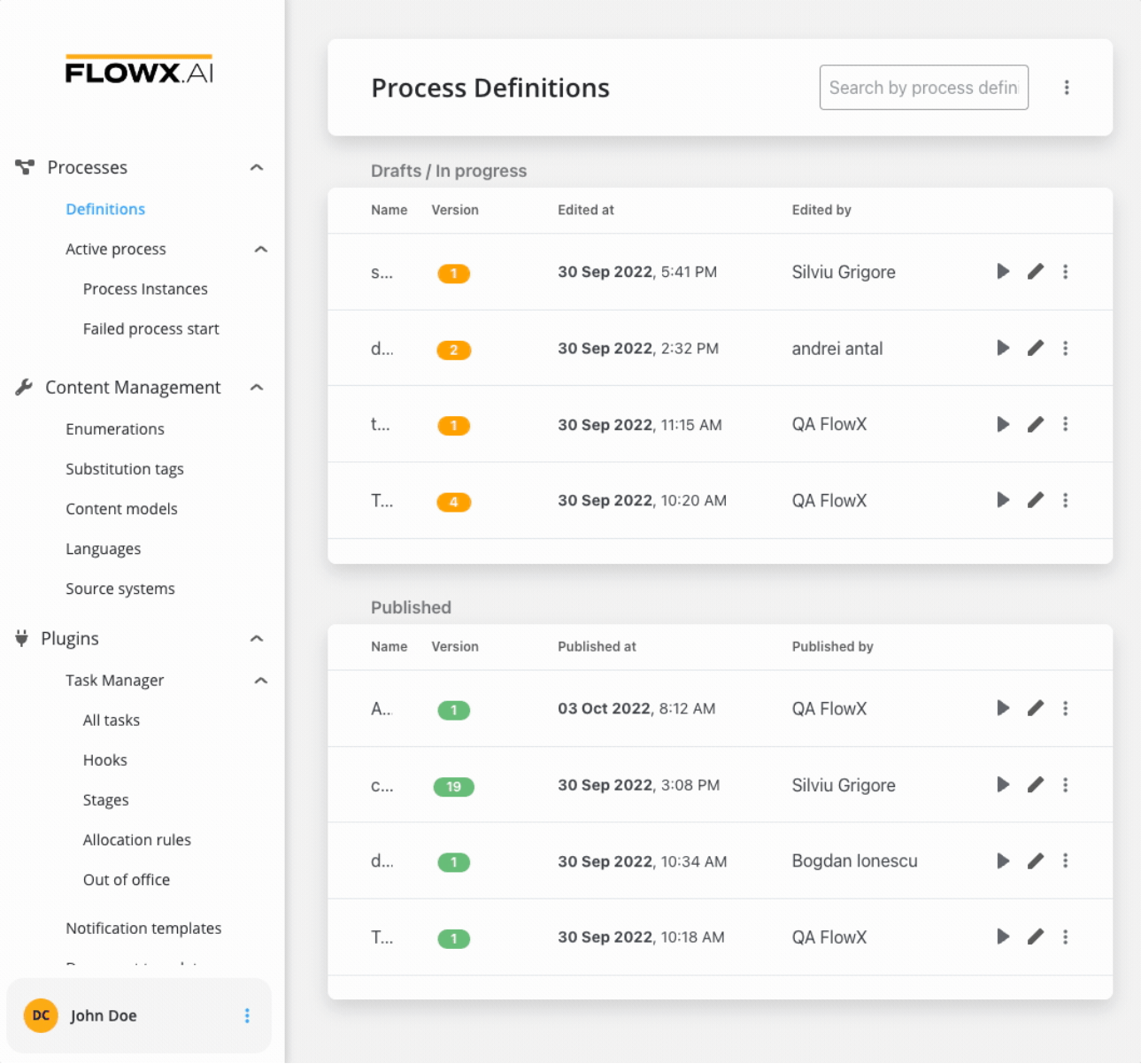
PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Generating docs based on templates / Generating from HTML templates

The Document Management Plugin allows you to generate documents based on previously defined document templates. This example specifically covers

generating documents using HTML templates.

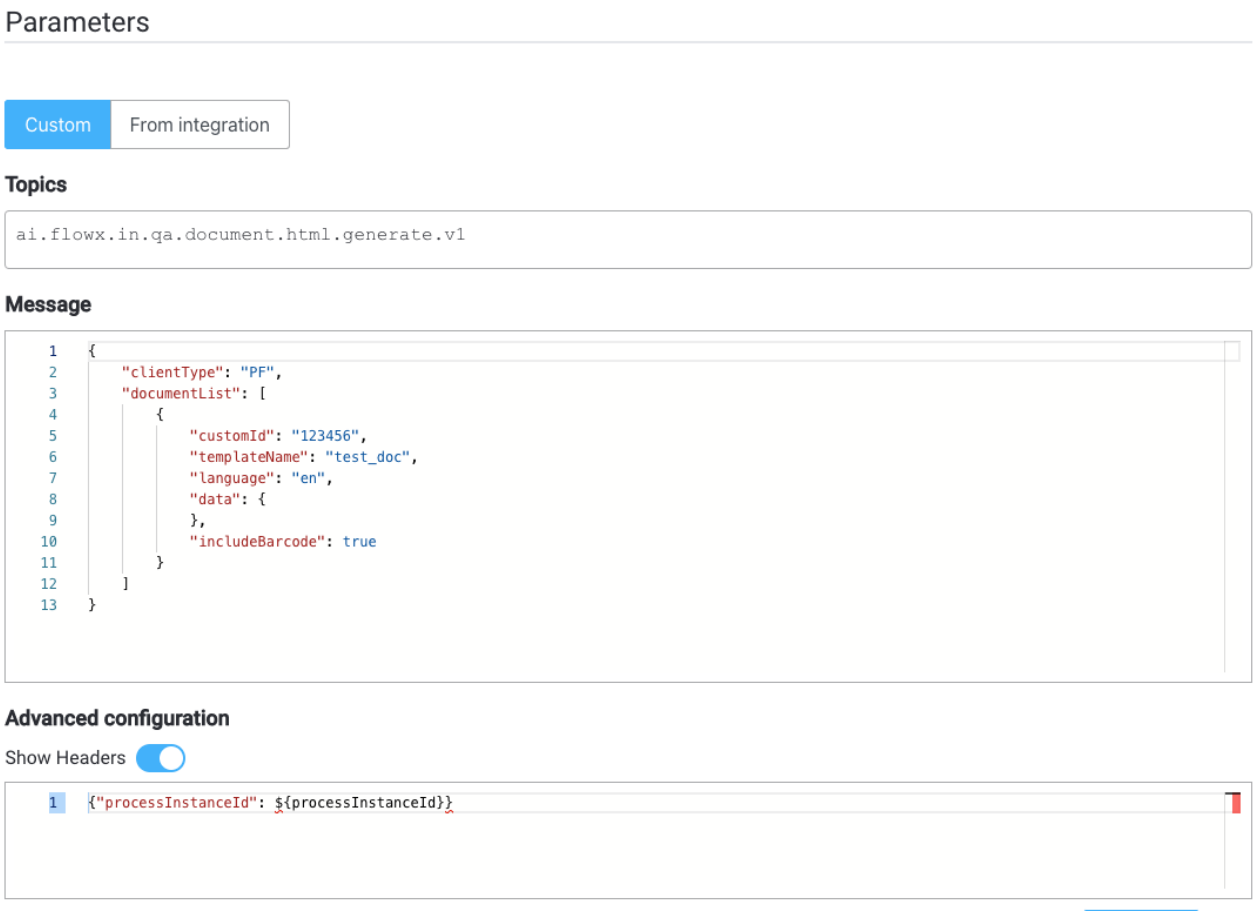
Creating a template

Use the WYSIWYG editor to create a document template.



Sending the request

1. Create a process that includes a **Kafka send event node** and a **Kafka receive event node** (one for sending the request and one for receiving the reply).
2. Configure the first node (Kafka Send Event) by adding a **Kafka send action**.
3. Add the **Kafka topic** to which the request should be sent.
4. Fill in the message with the following expected values in the request body:



- **documentList**: A list of documents to be generated with properties (name and value to be replaced in the document templates)

- **customId**: Client ID
- **templateName**: The name of the template to be used
- **language**
- **includeBarcode**: True/False
- **data**: A map containing the values that should be replaced in the document template. The keys used in the map should match the ones defined in the HTML template.

INFO

Kafka topic names can be set by using (overwriting) the following environment variables in the deployment:

- `KAFKA_TOPIC_DOCUMENT_GENERATE_HTML_IN` - default value: `ai.flowx.in.qa.document.html.generate.v1` - the topic that listens for the request from the engine
- `KAFKA_TOPIC_DOCUMENT_GENERATE_HTML_OUT` - default value: `ai.flowx.updates.qa.document.html.generate.v1` - the topic on which the engine expects the reply

The above examples of topics are extracted from an internal testing environment. When setting topics for other environments, follow the pattern `ai.flowx.updates.{{environment}}.document.generate.v1`.

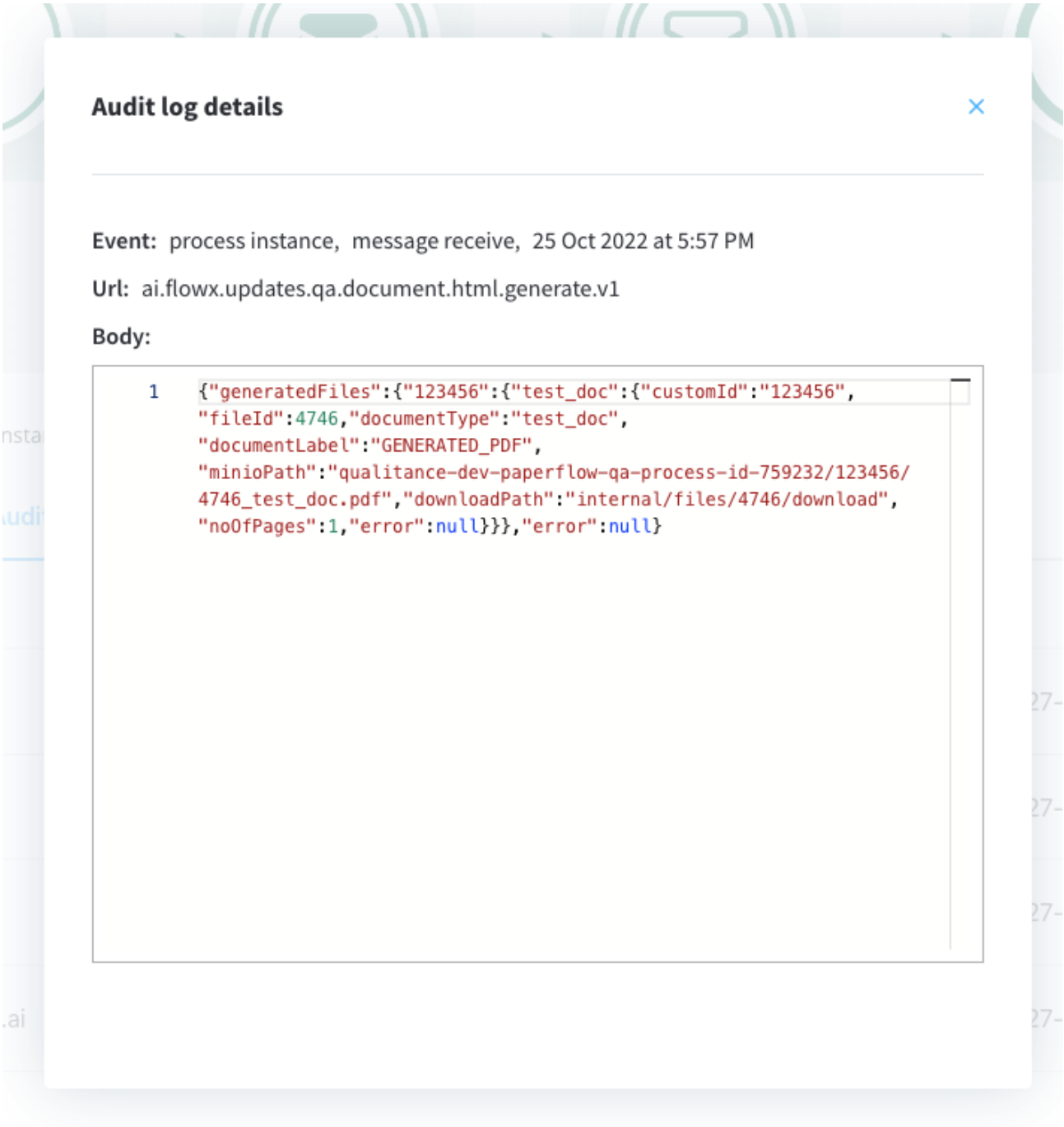
CAUTION

The engine listens for messages on topics with specific naming patterns. Make sure to use an outgoing topic name that matches the pattern configured in the engine.

Reply

> **INFO**
>
> You can view the response by accessing the **Audit log** menu.

The response will be sent on the output Kafka topic defined in the Kafka Receive Event Node. The response will contain the following information:



Values expected in the event body:

- **generatedFiles**: List of generated files.
  - **customId**: Client ID.

- **fileId**: The ID of the generated file.
- **documentType**: The name of the document template.
- **documentLabel**: A label or description for the document.
- **minioPath**: The path where the converted file is saved. It represents the location of the file in the storage system, whether it's a MinIO path or an S3 path, depending on the specific storage solution.
- **downloadPath**: The download path for the converted file. It specifies the location from where the file can be downloaded.
- **noOfPages**: The number of pages in the generated file.
- **error**: If there were any errors encountered during the generation process, they would be specified here. In the provided example, the value is null, indicating no errors.

Example of generated file response received on `KAFKA_TOPIC_DOCUMENT_GENERATE_HTML_IN` topic :

```
{
  "generatedFiles": {
    "123456": {
      "test_doc": {
        "customId": "123456",
        "fileId": 4746,
        "documentType": "test_doc",
        "documentLabel": "GENERATED_PDF",
        "minioPath": "qualitance-dev-paperflow-qa-process-
id-759232/123456/4746_test_doc.pdf", //or S3 path, depending
on your storage solution
        "downloadPath": "internal/files/4746/download",
        "noOfPages": 1,
        "error": null
```

```
        }
      }
    },
    "error": null
}
```


Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Generating docs based on templates / Managing HTML templates

In the Document Management Plugin, you have the flexibility to define and manage HTML templates for generating documents. These templates can incorporate various types of parameters to customize the content. Let's explore the different types of parameters and their specifications:

Configuring HTML templates

Text parameters

Text parameters are used to include dynamic text in the template. For example, you can include the company name and registration number in an offer document. Here's an example of HTML template specifications:

Lorem ipsum: Test Company SRL, dolor sit amet **RO1234567**.

```
<p><strong>Lorem ipsum: <span th:text="${companyName}">
</span></strong>, dolor sit amet <strong><span
th:text="${cui}"></span></strong>.</p>
```

Data specifications:

```
{
  "data": {
    "companyName": "Test Company SRL",
    "cui": "RO1234567"
  }
}
```

Dynamic tables - repeatable rows

Dynamic tables are useful when you want to display a table with repeatable rows. Each row can represent a different element from a generated list of objects. Here's an example of HTML template specifications:

Lorem ipsum The greatest offer - deluxe edition dolor sit amet, consectetur adipiscing elit. Nullam ante quam, dictum et accumsan quis, laoreet id lorem. Mauris bibendum consequat viverra. Ut accumsan volutpat augue. Cras id tortor hendrerit, fringilla ligula et, consequat quam. Proin quis dui et nisi ullamcorper pretium nec eu nulla. Sed ut sapien ac arcu accumsan varius. Proin faucibus augue tellus, at ultrices sapien vestibulum non. Nam pellentesque augue eu molestie sagittis.

Name	Value
Price (USD/MWh)*	25
Distribution rate (USD /MWh)**	C1 category: 27, C2 category: 29
Subscription price / day / place of consumption***	C1 category: 1.25, C2 category: 1.32
Period of validity of the price	Validity time fixed price Monday, from the start date of delivery to the date of completion of delivery
Payment term	90 days

```
<table>
    <thead>
        <tr class="headings">
            <th class="column-title">Name</th>
            <th class="column-title">Value</th>
        </tr>
    </thead>
    <tbody>
        <tr class='even pointer' th:each="row:
${offerValuesRows}" id="tablerow">
            <td th:each="header: ${offerValuesHeader}"
th:text="${row.get(header)}">
        </tr>
    </tbody>
</table>
```

Data specifications:

```
"data": {
   "offerValuesHeader": [
     "Name",
     "Value"
   ],
   "offerValuesRows": [
     { "Name": "Price (USD/MWh)", "Value": "25" },
     { "Name": "Distribution rate (USD/MWh)", "Value": "C1
category: 27, C2 category: 29" },
     { "Name": "Subscription price / day / place of
consumption", "Value": "C1 category: 1.25, C2 category:
1.32" },
     { "Name": "Period of validity of the price", "Value":
"Validity time fixed price Monday, from the start date of
delivery to the date of completion of delivery" },
     { "Name": "Payment term", "Value": "90 days" }
   ]
 }
```

Dynamic tables - repeatable table

This type of dynamic table allows you to display a table multiple times based on the elements of a generated list of objects. Here's an example of HTML template specifications:

Oferta Denumire oferta este aplicabila urmatoarelor locuri de consum:

Loc de consum	Distribuitor	Cod CLC	Modalitatate introducere consum	Tip consum	Categorie consum (MWh)	Consum total anual (MWh)
Lorem ipsum	Distribuitor 1	123456	Lorem ipsum kghf	Lorem ipsum	Lorem ipsum	Lorem ipsum

Loc de consum	Distribuitor	Cod CLC	Modalitatate introducere consum	Tip consum	Categorie consum (MWh)	Consum total anual (MWh)
Lorem ipsum	Distribuitor 2	131313	Lorem ipsum tryuty	Lorem ipsum	Lorem ipsum	Lorem ipsum

```
<p>Offer:</p>
<div th:each="type: ${consumptionPoints}">
<table>
    <thead>
       <tr>
          <th> Usage place </th>
          <th> Distributor </th>
          <th> CLC code </th>
          <th> Usage method input </th>
          <th> Usage type </th>
          <th> Usage category \n(MWh) </th>
          <th> Total usage \n(MWh) </th>
       </tr>
    </thead>
    <tbody>
       <tr th:if="${type.consumptionPoint.empty}">
           <td colspan=\"7\"> No information available here!
</td>
        </tr>
        <tr th:each=\"consumptionPoint :
${type.consumptionPoint}\=">
```

```
            <td><span
th:text="${consumptionPoint.consumptionPoint}"> Usage place
</span></td>
            <td><span
th:text="${consumptionPoint.distribuitor}"> Distributor
</span></td>
            <td><span th:text="${consumptionPoint.clcCode}">
Cod CLC </span></td>
            <td><span
th:text="${consumptionPoint.consumerInputMethod}"> Usage
method input </span></td>
            <td><span
th:text="${consumptionPoint.consumerType}"> Usage type
</span></td>
            <td><span
th:text="${consumptionPoint.consumerCategory}"> Usage
category \n(MWh) </span></td>
            <td><span
th:text="${consumptionPoint.totalAnnualConsumption}"> Total
usage \n(MWh) </span></td>
         </tr>
      </tbody>
   </table>
</div>
```

Data specifications:

```
  "data": {
    "consumptionPoints": [
      {
        "consumptionPoint": [
          {
            "consumptionPoint": "Lorem ipsum",
```

```
            "distribuitor": "Distributor 1",
            "clcCode": "123456",
            "consumerInputMethod": "Lorem ipsum",
            "consumerType": "Lorem ipsum",
            "consumerCategory": "Lorem ipsum",
            "totalAnnualConsumption": "Lorem ipsum"
          }
        ]
      },
      {
        "consumptionPoint": [
          {
            "consumptionPoint": "Lorem ipsum",
            "distribuitor": "Distributor 2",
            "clcCode": "131313",
            "consumerInputMethod": "Lorem ipsum ipsum",
            "consumerType": "Lorem ipsum",
            "consumerCategory": "Lorem ipsum",
            "totalAnnualConsumption": "Lorem ipsum"
          }
        ]
      }
    ]
  }
```


Dynamic sections

Dynamic sections allow you to display specific content based on certain conditions. For example, you can display a paragraph only when a certain condition is met. Here's an example of HTML template specifications:

PJ section, visible only if pjCLient = true

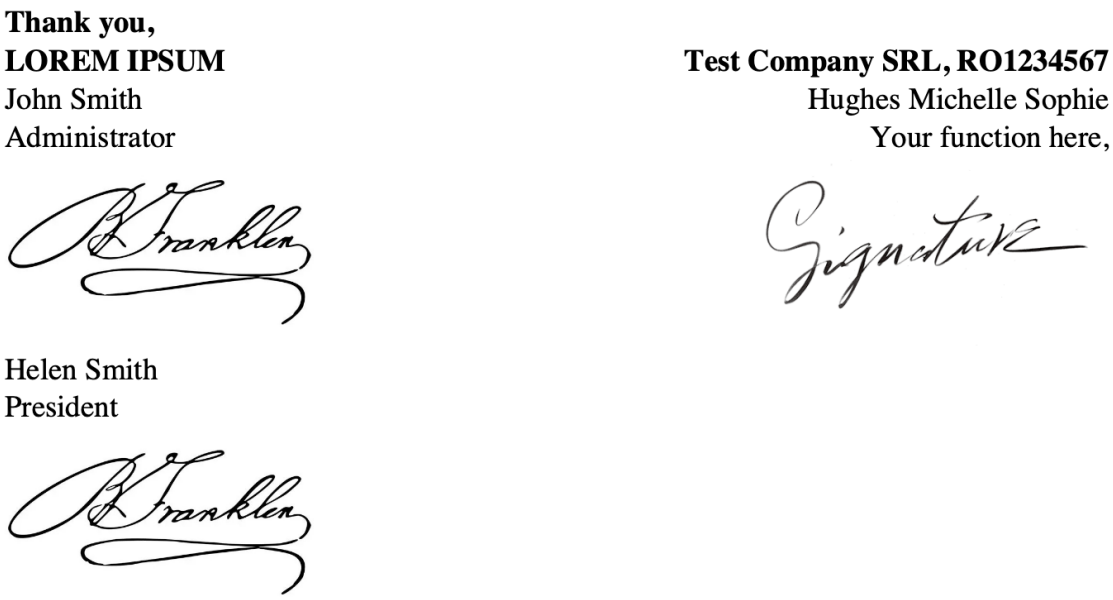
```
<span th:if="${pjCLient==true}">
    <p><b>PJ section, visible only if pjCLient = true</b>
</p>
    <p><span th:text="${termTechnicalServices}"></span></p>
</span>
<span th:if="${pjCLient==false}">
    <p><b>PF section, visible only if pjCLient = false</b>
</p>
    <p><span th:text="${termInsuranceServices}"></span></p>
</span>
```

Data specifications:

```
"data": {
   "pjCLient": true
 }
```

Images

You can include images in your final document by referencing them in the template. Here's an example of HTML template specifications:




```
<td class='align'><img th:src="*
{''+signature}" alt=\"\"
height='100px'/></td>
```

Data specifications:

```
"data": {
    "signature": "INSERT_BASE64_IMAGE"
  }
```

Barcodes

If you want to include a barcode, you can set the `includeBarcode` parameter to true.

For information on how to use barcodes and OCR, check the following section.

» OCR plugin

Lists

Lists are useful for displaying values from selected items in a checkbox as a bulleted list. Here's an example of HTML template specifications:

Income source:

- Income 1
- Income 2
- Income 3
- Income 4

```
  <div th:if="${incomeSource != null}">
    <h3>Income source:</h3>
    <ul>
       <li th:each="item : ${incomeSource}"
th:text="${item}"></li>
    </ul>
  </div>
```

Data specifications:

```
{
    "data": {
        "incomeSource": [
            "Income 1",
```

```
            "Income 2",
            "Income 3",
            "Income 4"
        ]
    }
}
```

Examples

> **TIP**
>
> Download a PDF sample generated based on the HTML example, **here**.

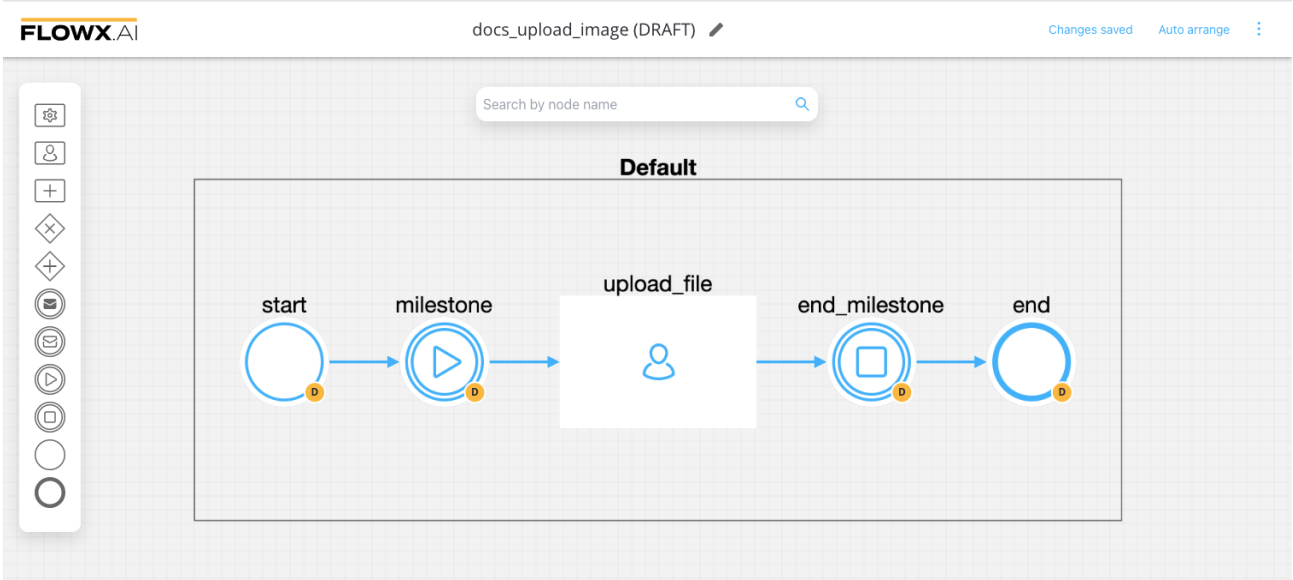
Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Uploading a new document

You can integrate document upload into a

The fallback content to display on prerendering
by adding a user task node with an **Upload action**. This allows users to interact with the process and choose which file to upload.

> **INFO**
>
> User task
>
> The fallback content to display on prerendering
> enable you to define and configure UI templates and actions for specific
> template config nodes, such as an upload file button.



To upload a document using a process, follow the next steps.

Defining the process

1. Create a process definition.
2. Add the necessary nodes, including **start/end nodes**, **start/end milestone nodes**, and a **user task node**.
3. Configure the user task node:
    - Configure the node settings.
    - Configure the upload action, including topics, document type, and folder.

- (UI) Configure the upload button.

Configuring the process definition

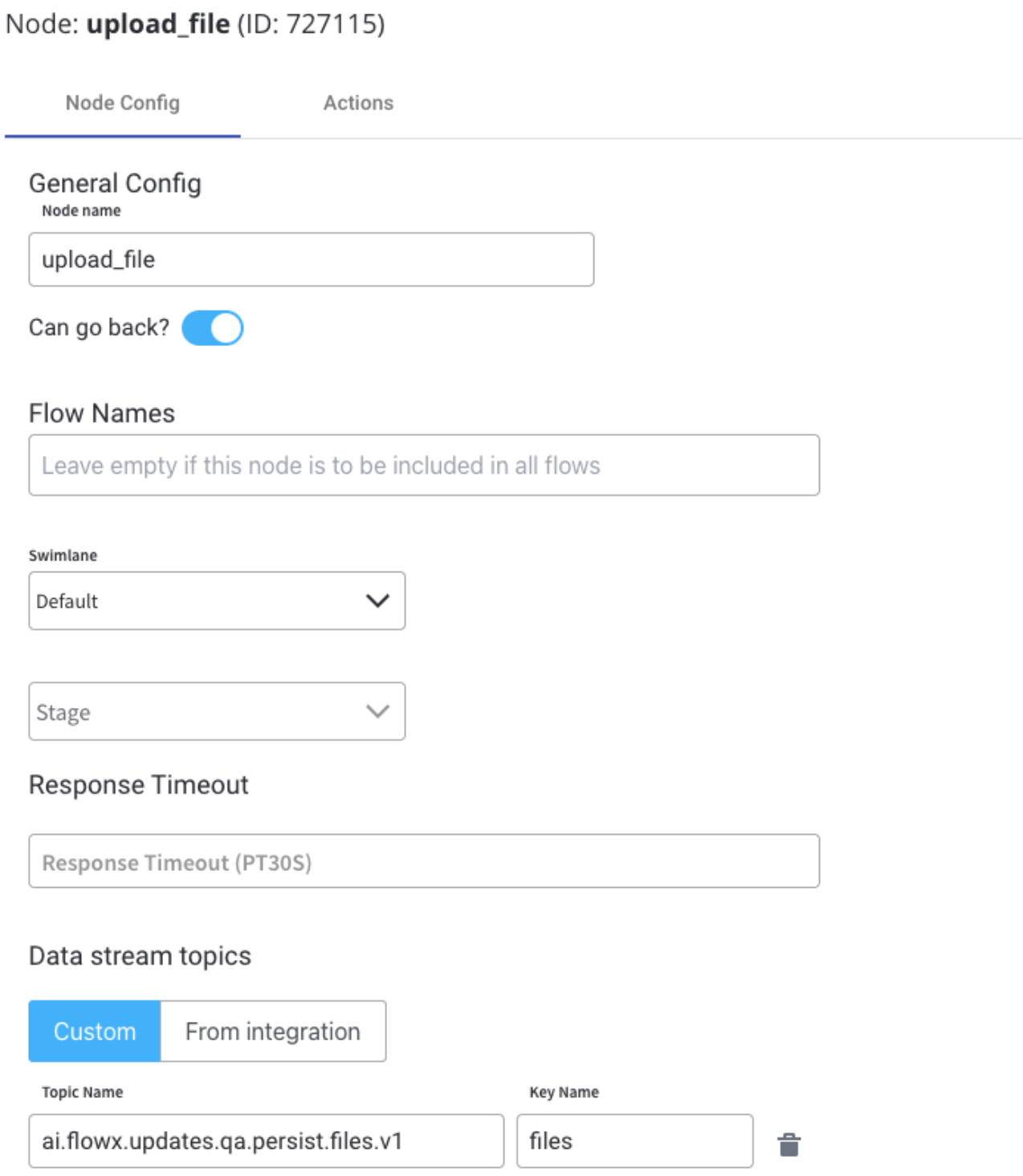
User task node

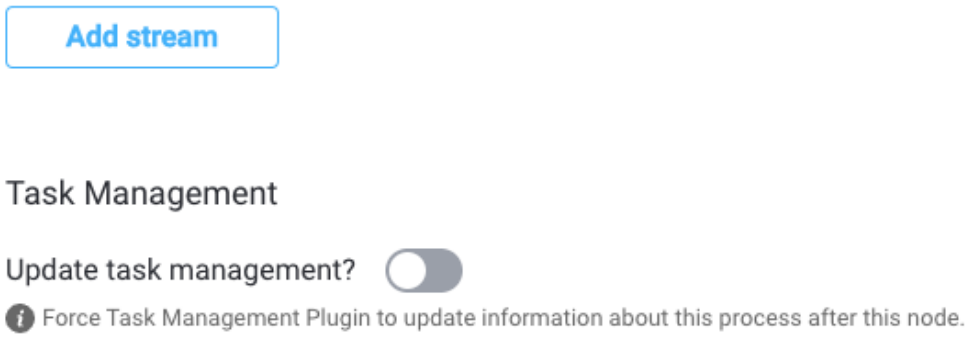
Node Config

- **Swimlane**: Choose a swimlane (if there are multiple swimlanes in the process) to restrict access to specific user roles. If there's only one swimlane, the value is "Default".
- **Stage**: Assign a stage to the node.
- **Topic Name**: Specify the topic name where the process engine listens for the response. This topic should be added to the platform and match the topic naming rule for the engine to listen to it. The default value is `ai.flowx.updates.qa.persist.files.v1`, extracted from `KAFKA_TOPIC_DOCUMENT_PERSIST_IN`.

> **CAUTION**
>
> A naming pattern must be defined in the
>
> The fallback content to display on prerendering
> configuration to use the specified topics. It's important to ensure that all events starting with the configured pattern are consumed by the Engine. For example, the `KAFKA_TOPIC_PATTERN` is the topic name pattern where the Engine listens for incoming
> The fallback content to display on prerendering
> events.

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

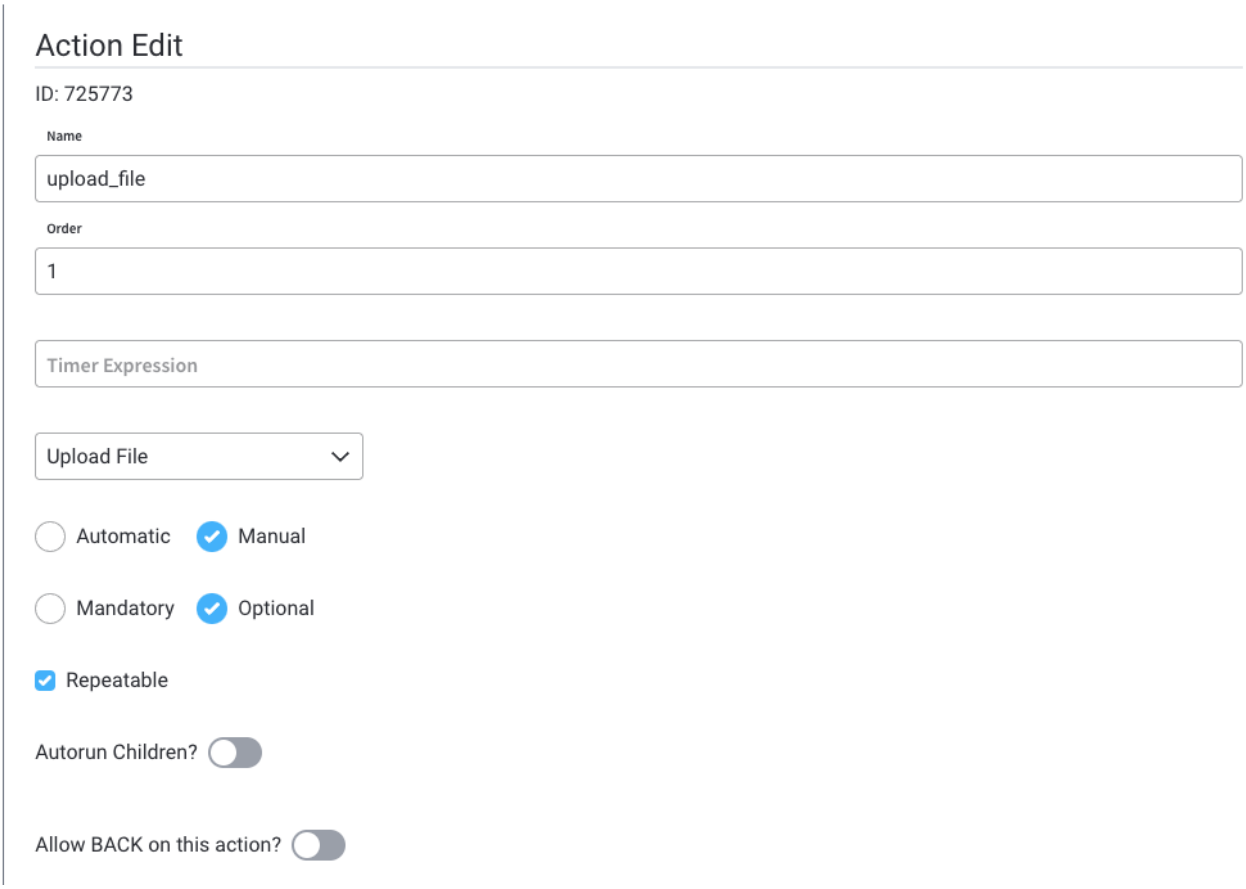




Actions

Actions edit

- **Action Type**: Set it to Upload File.
- **Trigger Type**: Choose Manual to allow user-triggered action.
- **Required Type**: Set it as Optional.
- **Reputable**: Check this option if the action can be triggered multiple times.
- **Autorun Children**: When enabled, the child actions defined as mandatory and automatic will run immediately after the parent action is finalized.



Parameters

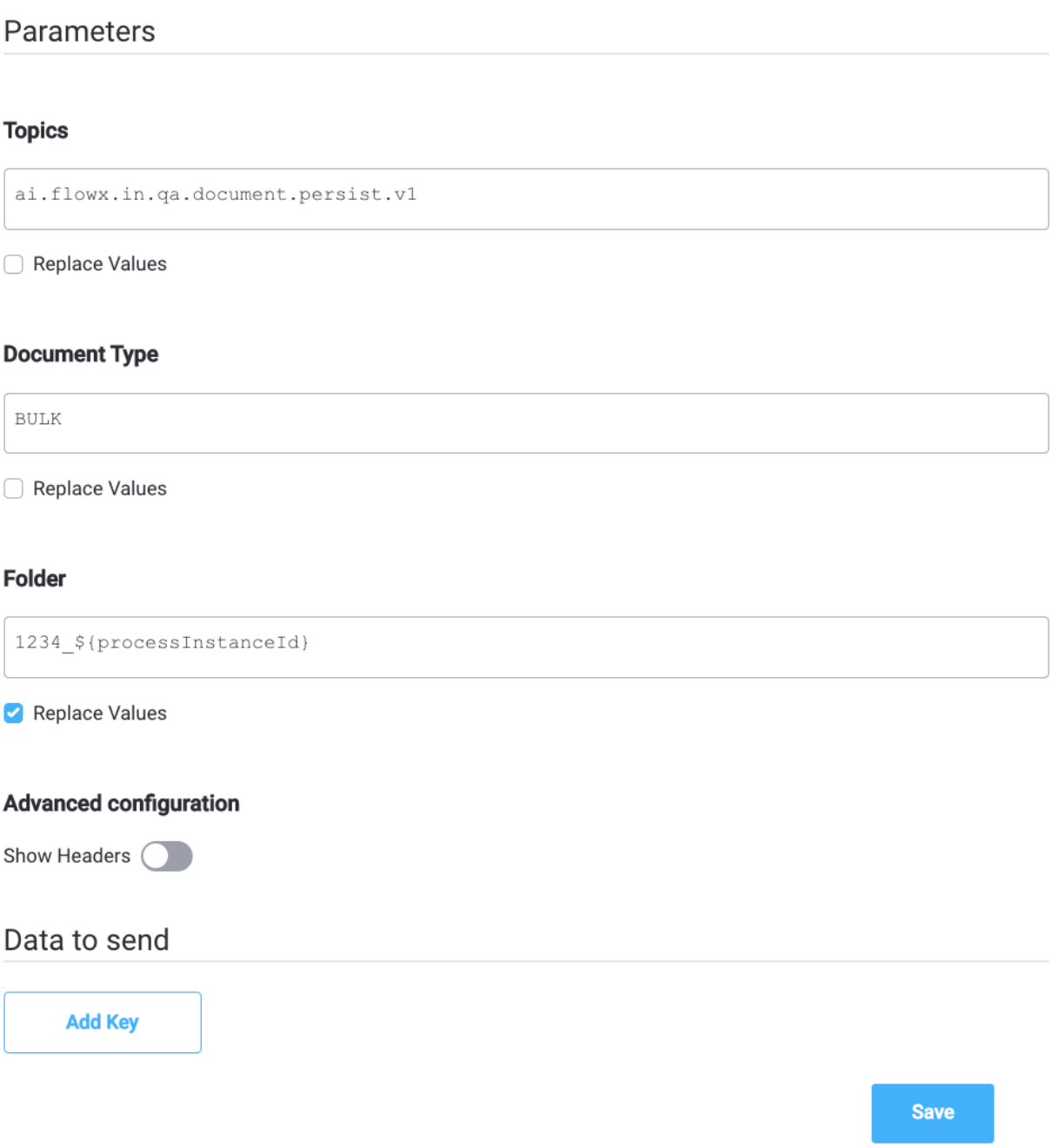
- **Topics**: Set it to `ai.flowx.in.document.persist.v1`, extracted from `KAFKA_TOPIC_DOCUMENT_PERSIST_IN`.
- **Document Type**: Set it to BULK.
- **Folder**: Allows you to configure a value by which the file will be identified in the future.
- **Advanced Configuration (Show Headers)**: Represents a JSON value that will be sent in the headers of the Kafka message.

> **INFO**

Kafka topic names can be customized by overwriting the following environment variables during deployment:

- `KAFKA_TOPIC_DOCUMENT_PERSIST_IN` - default value: `ai.flowx.in.qa.document.persist.v1`
- `KAFKA_TOPIC_DOCUMENT_PERSIST_OUT` - default value: `ai.flowx.updates.qa.document.persist.v1`

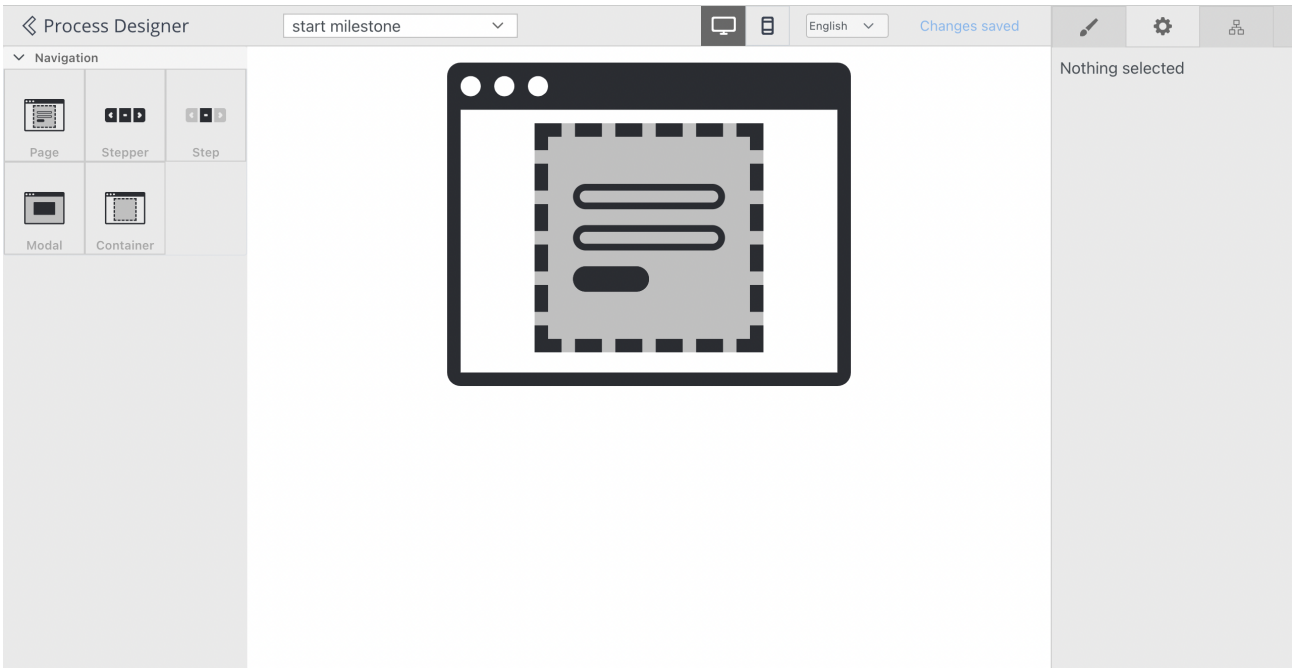
The above examples of topics are extracted from an internal testing environment. When setting topics for other environments, follow this pattern: `ai.flowx.updates.{{environment}}.document.persist.v1`.



Milestone nodes

You can configure start and end milestone nodes before and after the user task. Additionally, you can add a modal template (e.g., a **Page**) to the start milestone

node to display a modal screen, as shown in the example above.

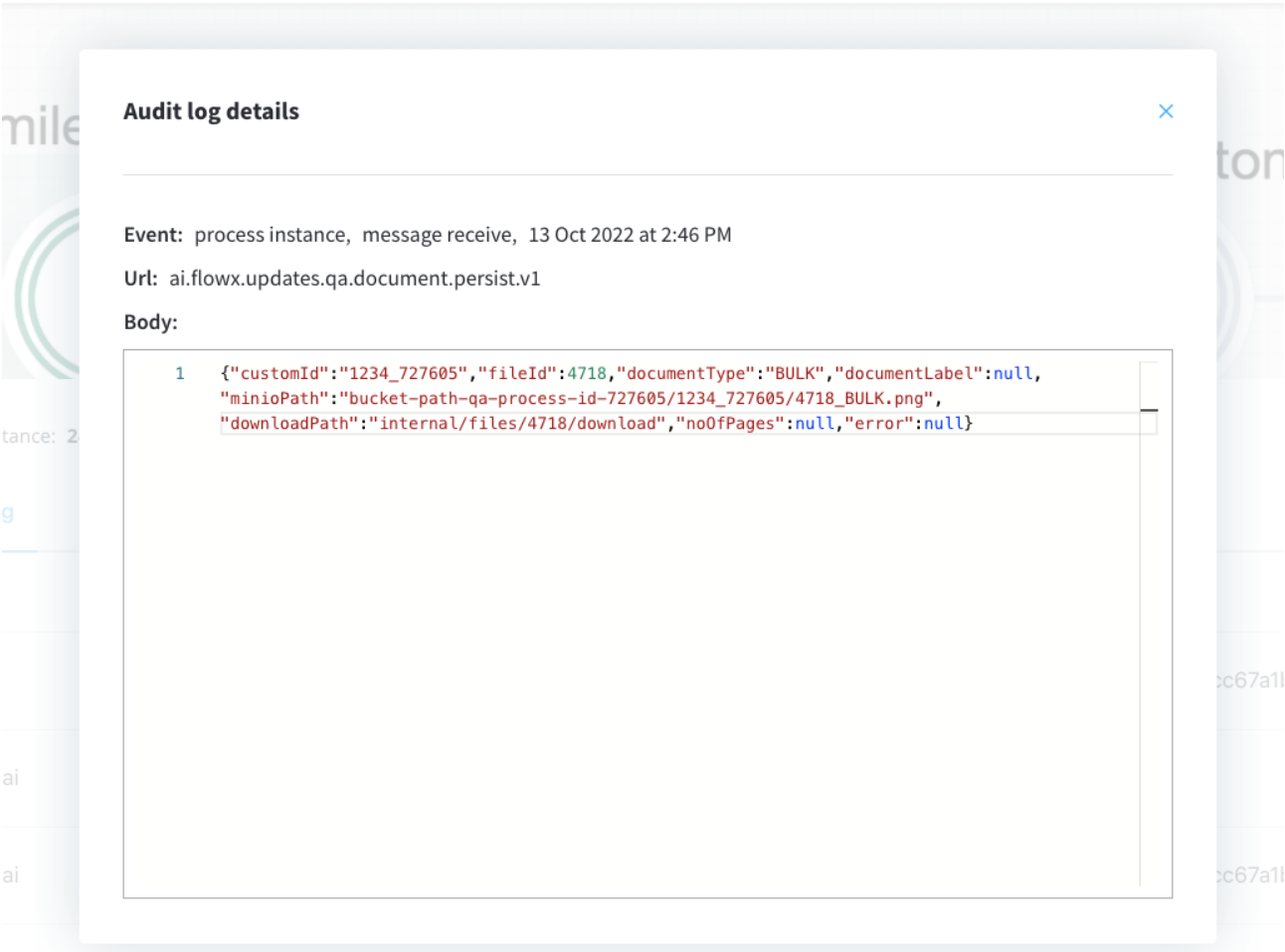


Receiving the reply

The reply body is expected to contain the following values:

- **customId**: The client ID.
- **fileId**: The ID of the file.
- **documentType**: The document type.
- **minioPath**: The path where the uploaded file is saved. It represents the location of the file in the storage system, whether it's a MinIO path or an S3 path, depending on the specific storage solution.
- **downloadPath**: The download path for the uploaded file. It specifies the location from where the file can be downloaded.
- **noOfPages**: The number of pages in the document.

> **INFO**
>
> You can view the response by accessing the **Audit log** menu.




```
{
    "customId" : "1234_727605",
    "fileId" : 4718,
    "documentType" : "BULK",
    "documentLabel" : null,
    "minioPath" : "bucket-path-qa-process-id-
```

```
727605/1234_726254/4718_BULK.png",
        "downloadPath" : "internal/files/4714/download",
        "noOfPages" : null,
        "error" : null
    }
```


Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Converting documents to different formats

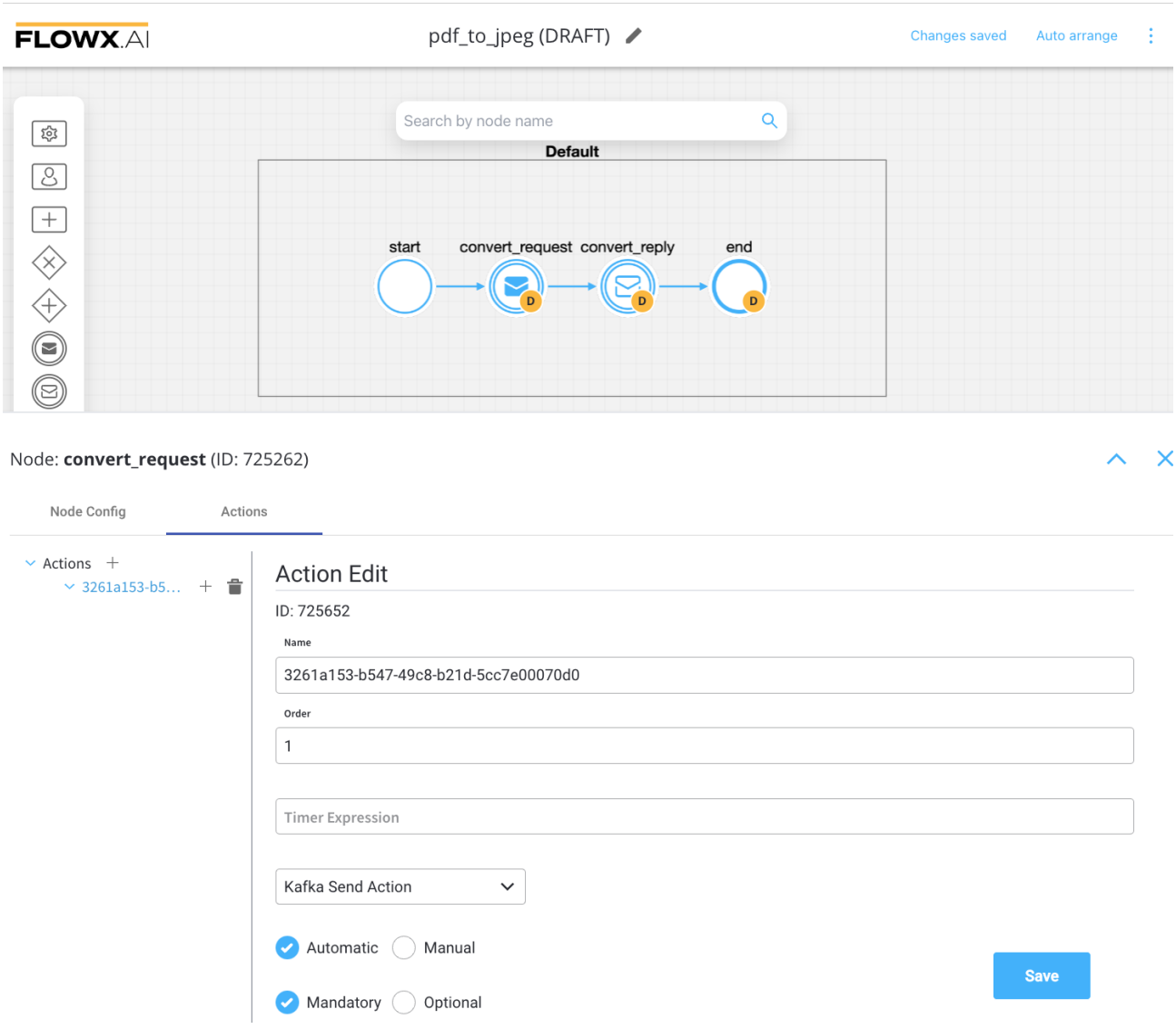
> ⚠ **CAUTION**
>
> Currently, the supported conversion method is from **PDF** to **JPEG**.

Sending the request

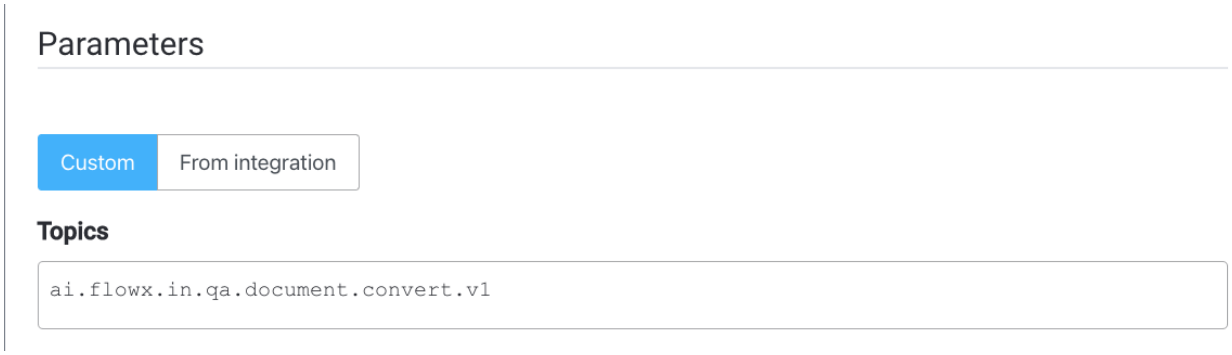
To create a process that converts a document from PDF to JPEG format, follow these steps:

1. Create a process that includes a **Kafka send event** node and a **Kafka receive event** node. The **send node** is used to send the conversion request, and the **receive node** is used to receive the reply.

2. Configure the first node (**Kafka send event**) by adding a **Kafka send action**. Here is an example:



3. Specify the **Kafka topic** where you want to send the conversion request:



4. Fill in the body of the message request:

- `fileId`: The file ID that will be converted
- `to`: The file extension to convert to (in this case, "jpeg").

> **INFO**
>
> You can set the Kafka topic names by overwriting the following environment variables during deployment:
>
> - `KAFKA_TOPIC_FILE_CONVERT_IN` - default value: `ai.flowx.in.qa.document.convert.v1` - the topic that listens for conversion requests from the engine

- `KAFKA_TOPIC_FILE_CONVERT_OUT` - default value: `ai.flowx.updates.qa.document.convert.v1` - the topic on which the engine expects the reply

The examples provided above are extracted from an internal testing environment. When setting topics for other environments, use the pattern `ai.flowx.updates.{{environment}}.document.convert.v1`.

CAUTION

Make sure to use an outgoing topic name for the reply that matches the pattern configured in the

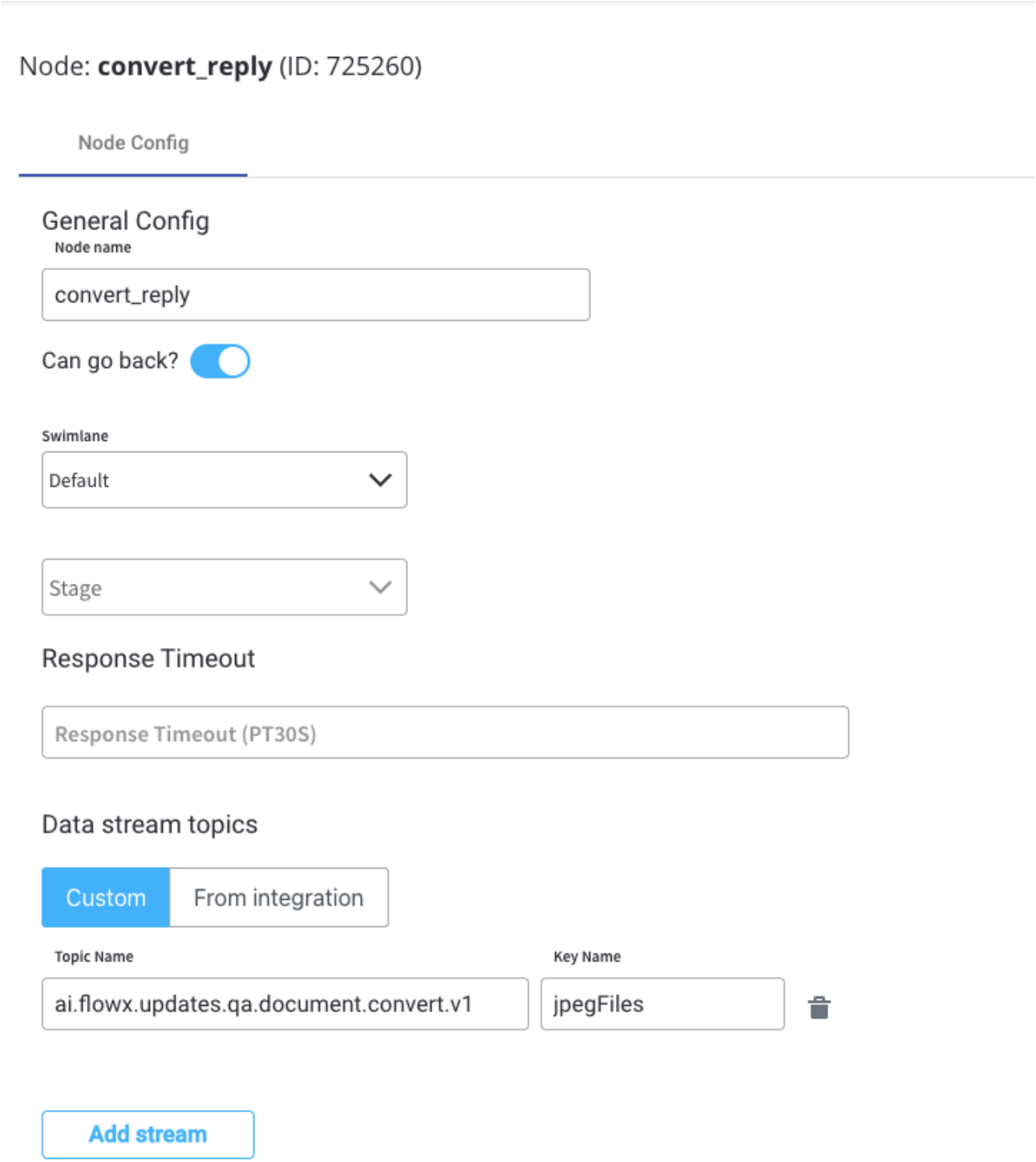
The fallback content to display on prerendering
, as it listens for messages on topics with specific names.

Receiving the reply

INFO

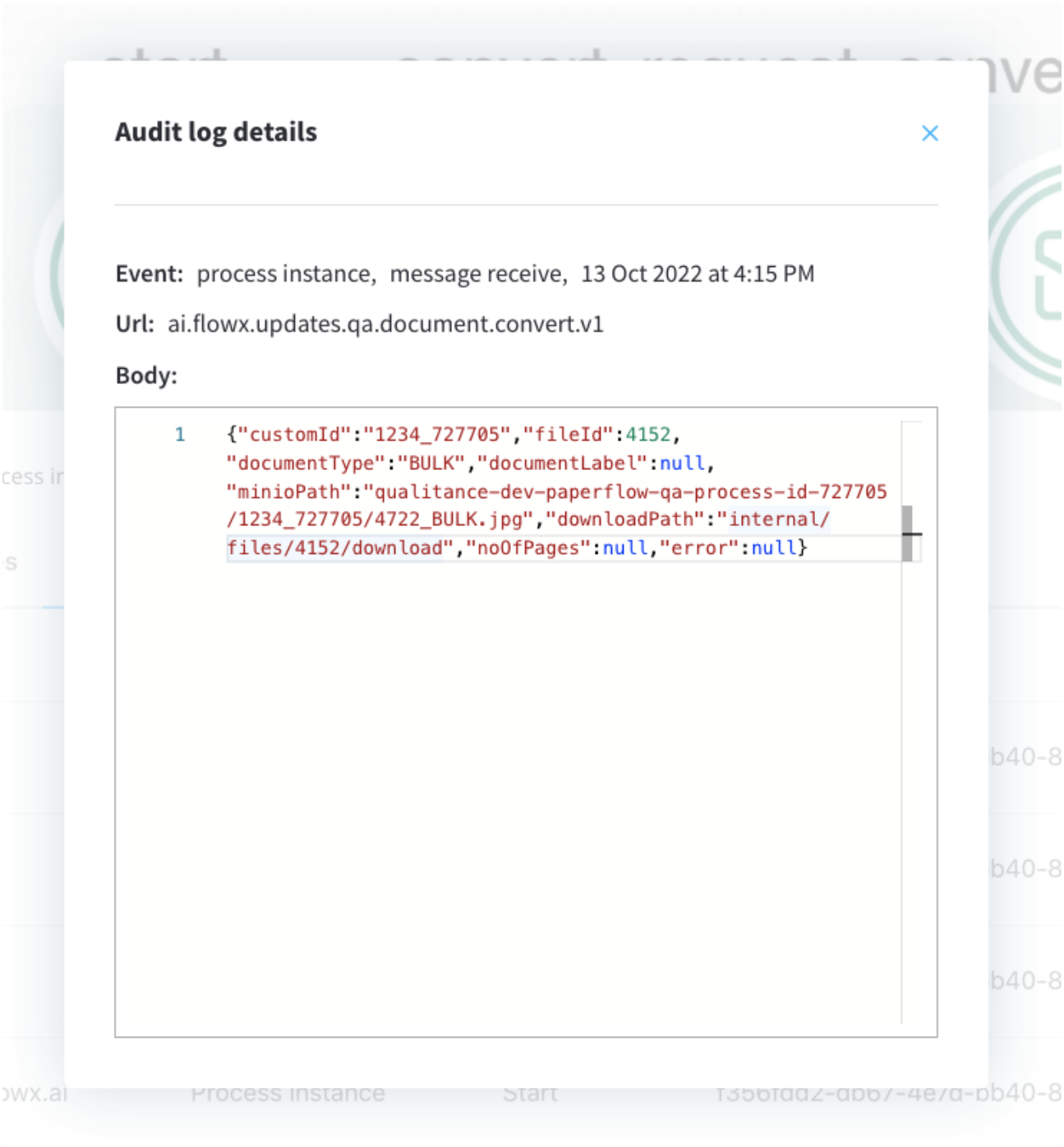
You can view the response by accessing the **Audit log** menu.

The response will be sent to the outgoing Kafka topic (defined on the Kafka receive event node) and can be accessed as follows:



Values expected in the reply body:

- **customId**: The client ID.
- **fileId**: The file ID.
- **documentType**: The document type.
- **documentLabel**: The document label (if available).
- **minioPath**: The path where the converted file is saved. It represents the location of the file in the storage system, whether it's a MinIO path or an S3 path, depending on the specific storage solution.
- **downloadPath**: The download path for the converted file.
- **noOfPages**: The number of pages in the converted file (if available).
- **error**: Any error message in case of an error during the conversion process.



Response:

```
{
  "customId": "1234_727705",
```

```
  "fileId": 4152,
  "documentType": "BULK",
  "documentLabel": null,
  "minioPath": "qualitance-dev-paperflow-qa-process-id-
727705/1234_727705/4152_BULK.jpg",
  "downloadPath": "internal/files/4152/download",
  "noOfPages": null,
  "error": null
}
```


Please note that the actual values in the response will depend on the specific conversion request and the document being converted.

Was this page helpful?

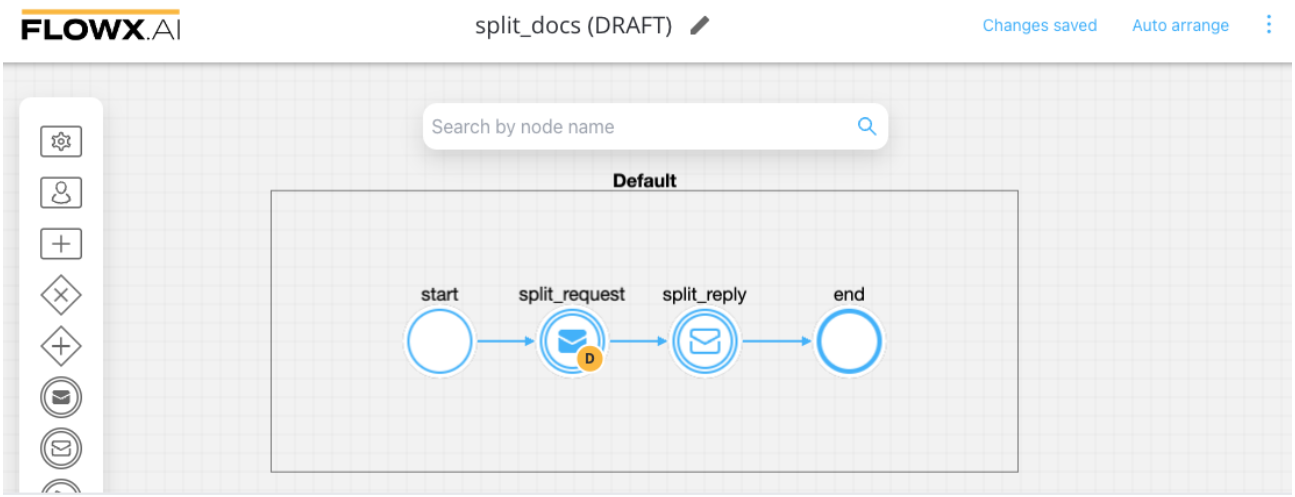
PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Splitting a document

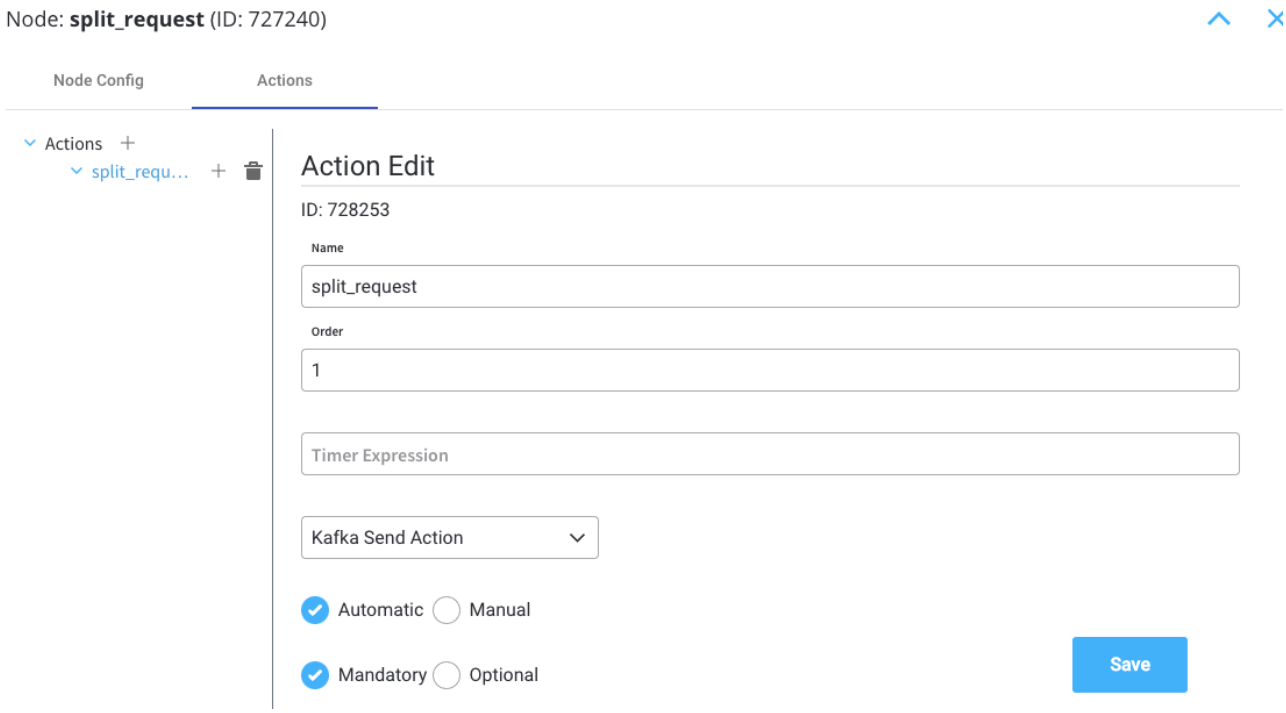
You can split a document into multiple parts using the Documents Plugin. This feature is useful, for example, when a user uploads a bulk scanned file that needs to be separated into separate files.

Sending the request

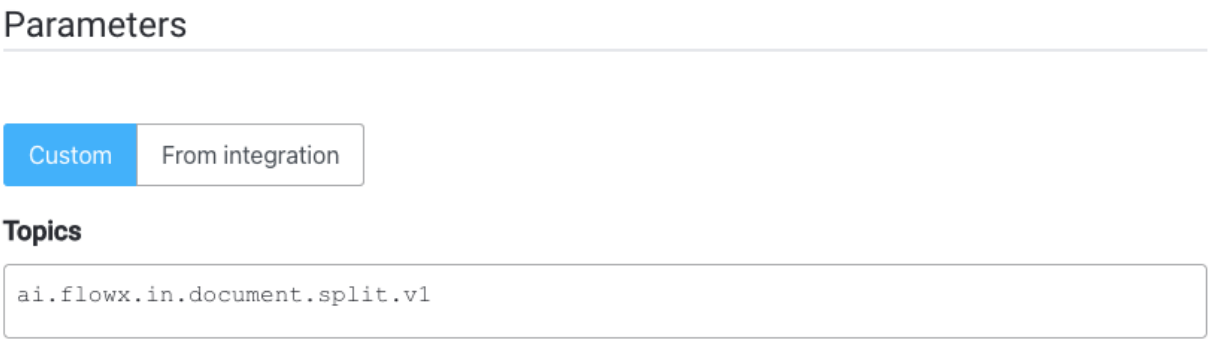
To split a document, follow these steps:

1. Create a process and add a **Kafka send event node** and a **Kafka receive event node**. These nodes are used to send the request and receive the reply.
2. Configure the first node, Kafka send event node by adding a **Kafka send action**.





3. Specify the **Kafka topic** to which you want to send the request.



4. Fill in the body message request:

- **fileId**: The ID of the file to be split.
- **parts**: A list containing information about the expected document parts.
  - **documentType**: The document type.
  - **customId**: The client ID.

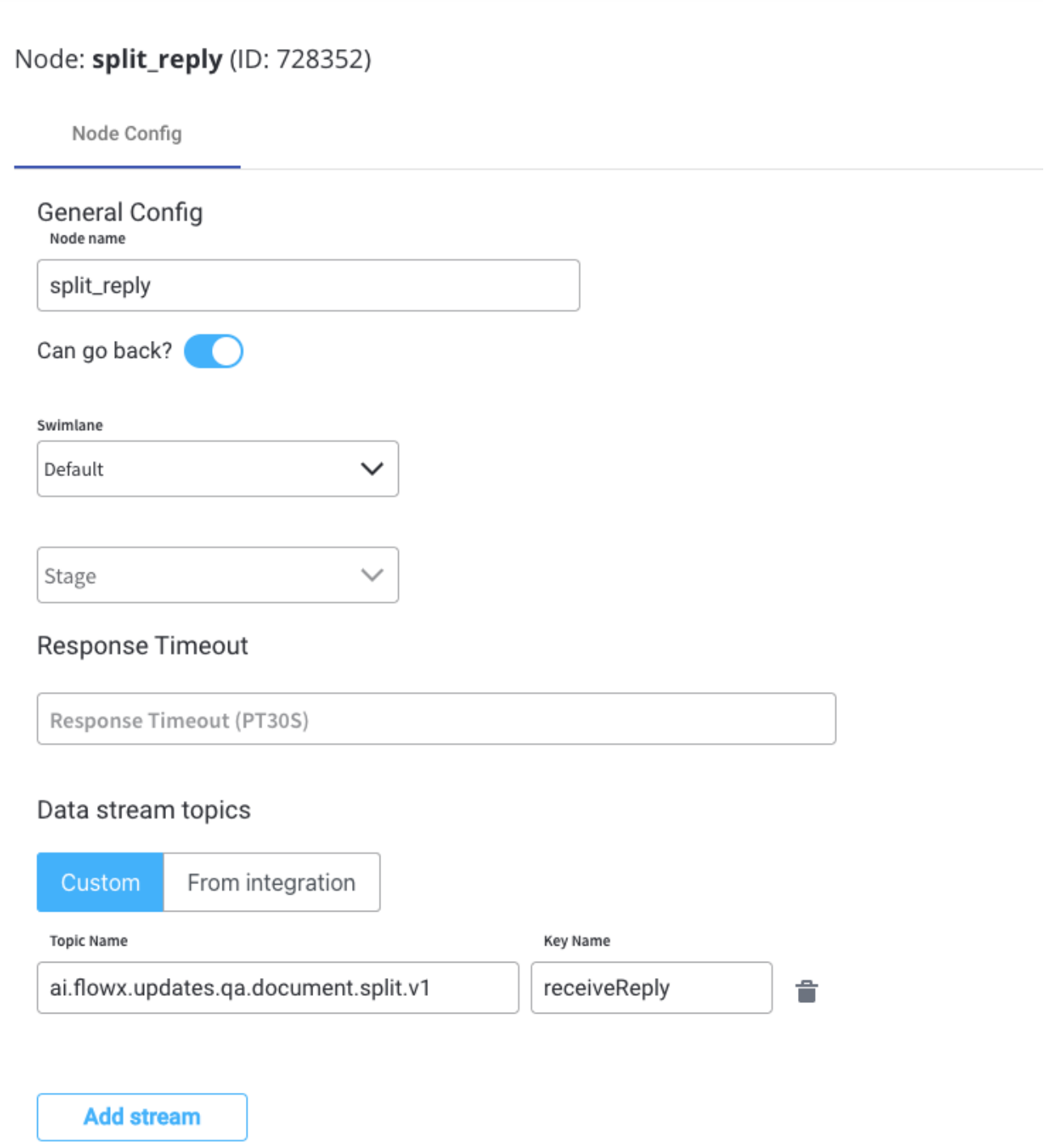
- **shouldOverride**: A boolean value (true or false) indicating whether to override an existing document if one with the same name already exists.
- **pagesNo**: The pages that you want to separate from the document.

> **INFO**
>
> You can customize the Kafka topic names by overwriting the following environment variables during deployment:
>
> `KAFKA_TOPIC_DOCUMENT_SPLIT_IN` - default value: `ai.flowx.in.qa.document.split.v1` - this is the topic that listens for the request from the engine
>
> `KAFKA_TOPIC_DOCUMENT_SPLIT_OUT` - default value: `ai.flowx.updates.qa.document.split.v1` - this is the topic on which the engine expects the reply
>
> The above examples of topics are extracted from an internal testing environment. When setting topics for other environments, follow this pattern: `ai.flowx.updates.{{environment}}.document.split.v1`.

> **CAUTION**
>
> The Engine listens for messages on topics with specific names. Make sure to use an outgoing topic name that matches the pattern configured in the Engine.

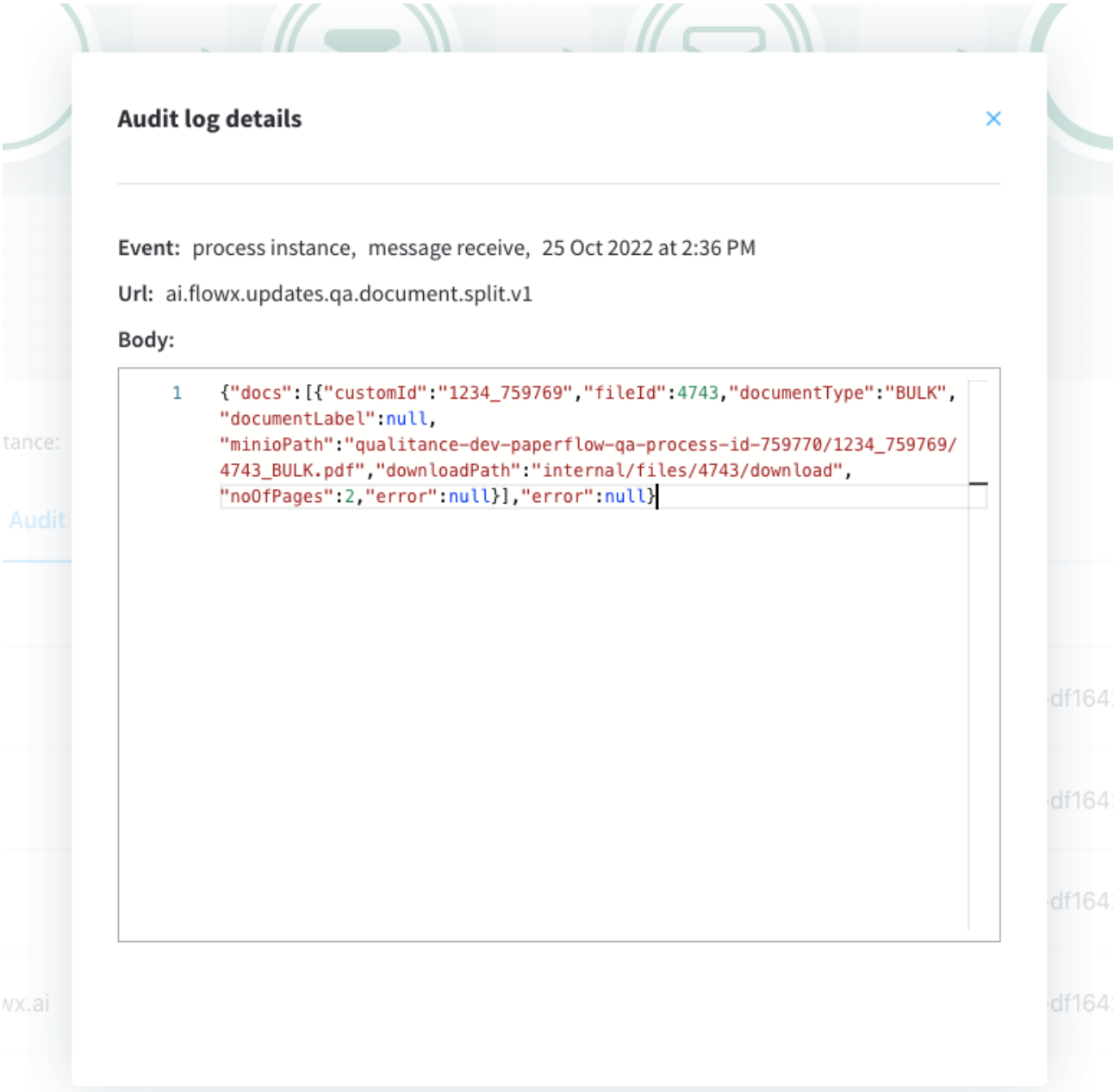
Receiving the reply

You can view the response by accessing the Audit log menu. The reply will be sent to the Kafka topic specified in the Kafka receive event node.



The response body will contain the following values:

- **docs**: A list of documents.
  - **customId**: The client ID.
  - **fileId**: The ID of the file.
  - **documentType**: The document type.
  - **minioPath**: The storage path for the document.
  - **downloadPath**: The download path for the document.
  - **noOfPages**: The number of pages in the document.



Here's an example of the response JSON:

```
{
  "docs": [
    {
```

```
      "customId": "1234_759769",
      "fileId": 4743,
      "documentType": "BULK",
      "documentLabel": null,
      "minioPath": "qualitance-dev-paperflow-qa-process-id-
759770/1234_759769/4743_BULK.pdf",
      "downloadPath": "internal/files/4743/download",
      "noOfPages": 2,
      "error": null
    }
  ],
  "error": null
}
```


Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Updating and deleting document files

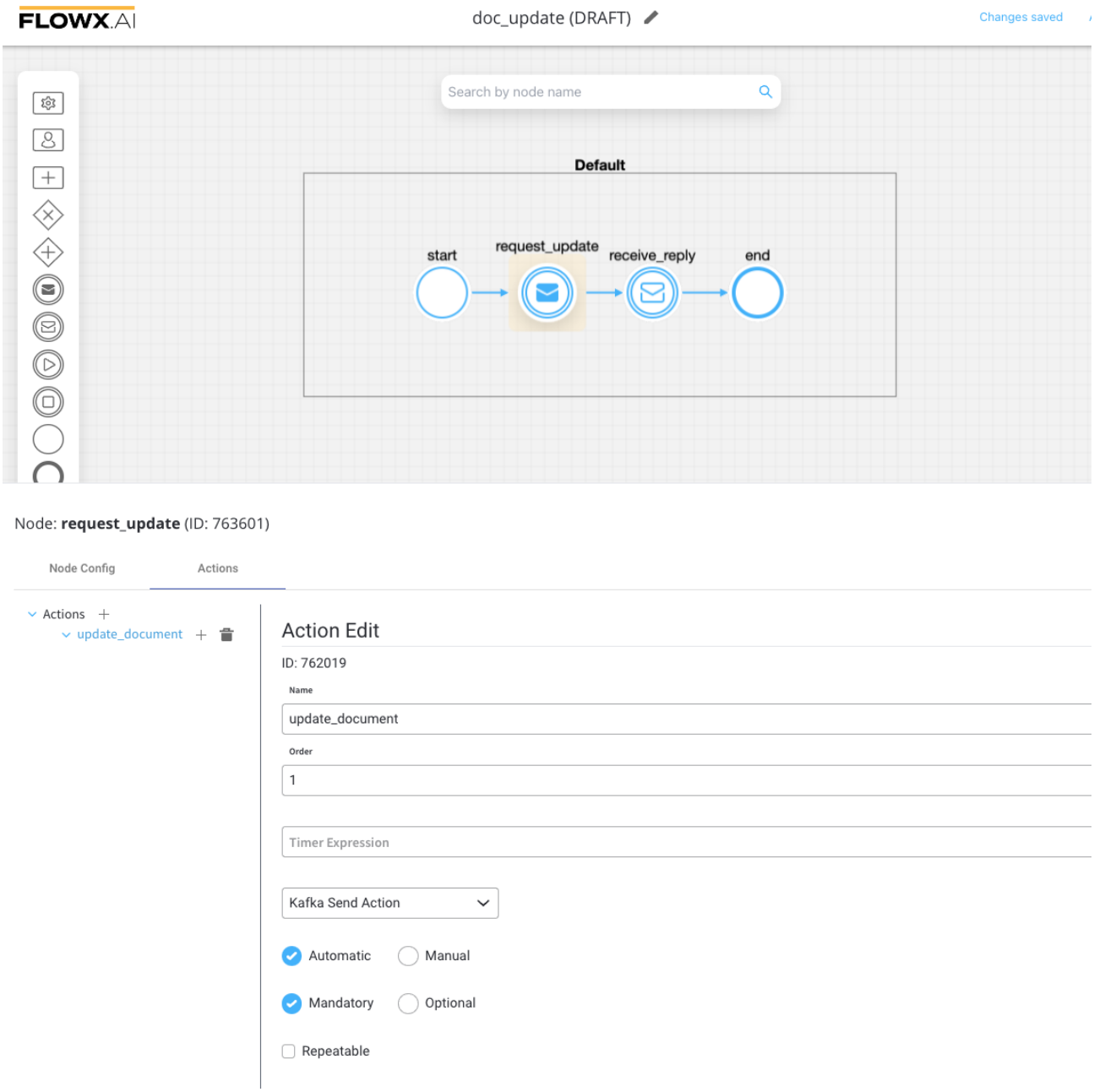
The documents plugin provides functionality for updating and deleting files associated with documents. You can update existing files or remove them from a document.

Updating files

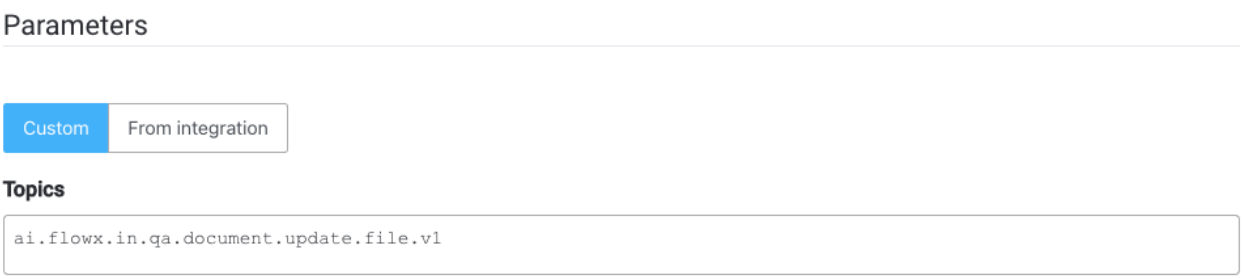
Sending the request

To update files, follow these steps:

1. Create a process and add a **Kafka send event node** and a **Kafka receive event node** (one for sending the request and one for receiving the reply).
2. Configure the first node (Kafka send event) by adding a **Kafka send action**.



3.Specify the **Kafka topic** to send the request to.



4. Fill in the body of the request message:

- **fileId**: The ID of the file.
- **customId**: The client ID.
- **documentType**: The document type.

> **INFO**

Kafka topic names can be customized by overwriting the following environment variables during deployment:

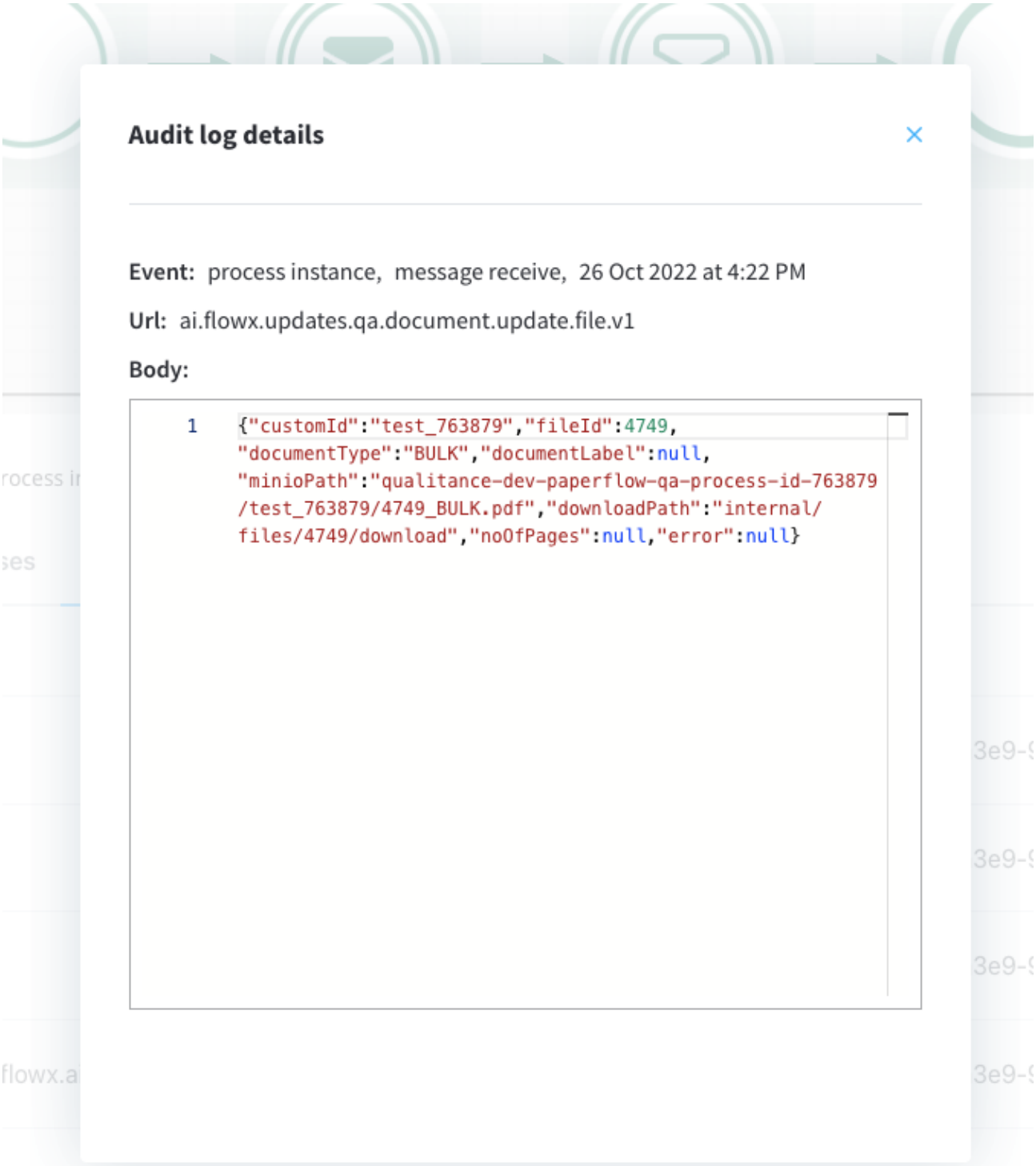
- `KAFKA_TOPIC_FILE_UPDATE_IN:` - default value: `ai.flowx.in.qa.document.update.file.v1`
- `KAFKA_TOPIC_FILE_UPDATE_OUT` - default value: `ai.flowx.updates.qa.document.update.file.v1`

The above examples of topics are extracted from an internal testing environment, when setting topics for other environments, follow the next pattern, for example, `ai.flowx.updates.{{environment}}.document.update.file.v1`.

CAUTION

Make sure to use an outgoing topic name that matches the pattern configured in the Engine, as the Engine listens for messages on topics with specific naming patterns.

Receiving the reply



Values expected in the reply body:

- customId = client ID
- fileId = file ID
- documentType = document type
- documentLabel = document label
- minioPath = minio path for the updated file
- downloadPath = download path for the updated file
- error = error description

Example:

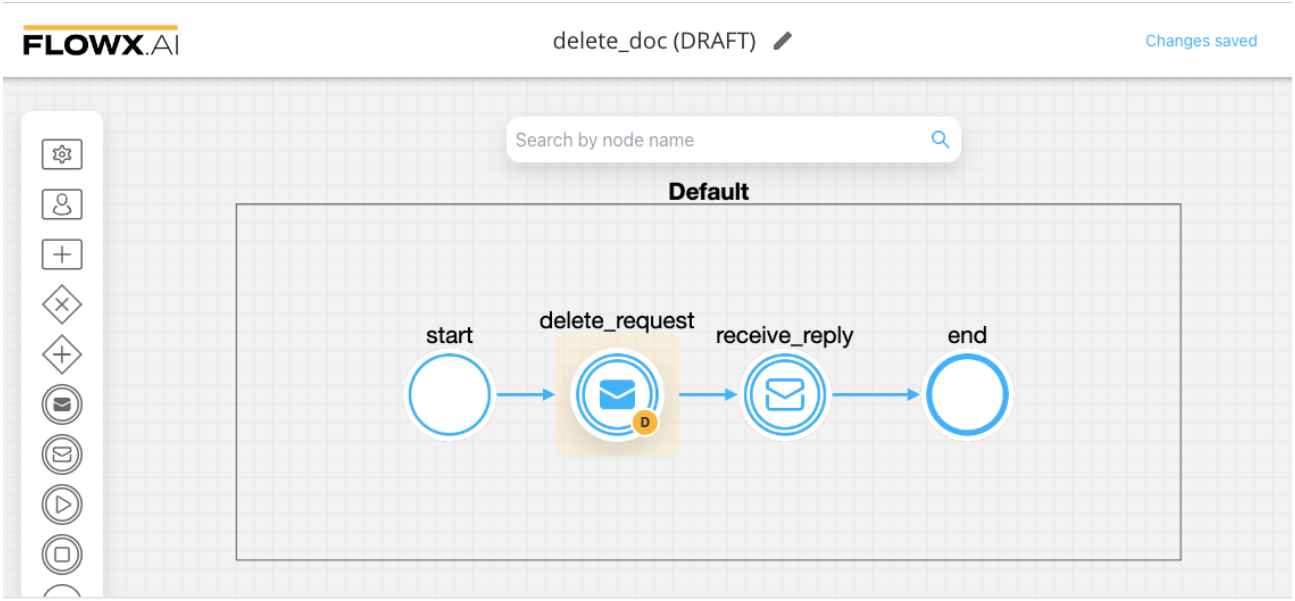
```
{
  "customId": "test_763879",
  "fileId": 4749,
  "documentType": "BULK",
  "documentLabel": null,
  "minioPath": "qualitance-dev-paperflow-qa-process-id-
763879/test_763879/4749_BULK.pdf",
  "downloadPath": "internal/files/4749/download",
  "noOfPages": null,
  "error": null
}
```

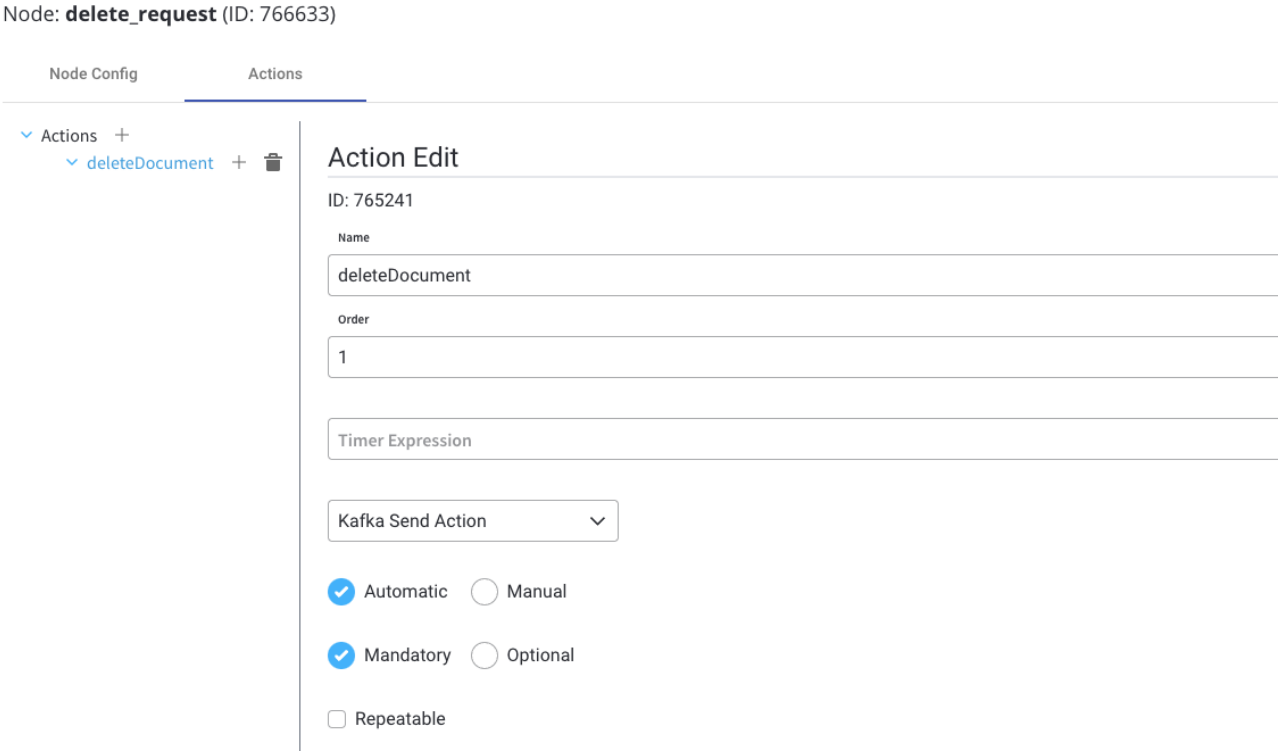
Deleting files from a document

Used to delete files after bulk upload.

Sending the request

1. Create a process in which you add a **Kafka send event node** and a **Kafka receive event node** (one to send the request, one to receive the reply).
2. Configure the first node (Kafka send event) - add a **Kafka send action**.





3. Add the **Kafka topic** where to send the request:

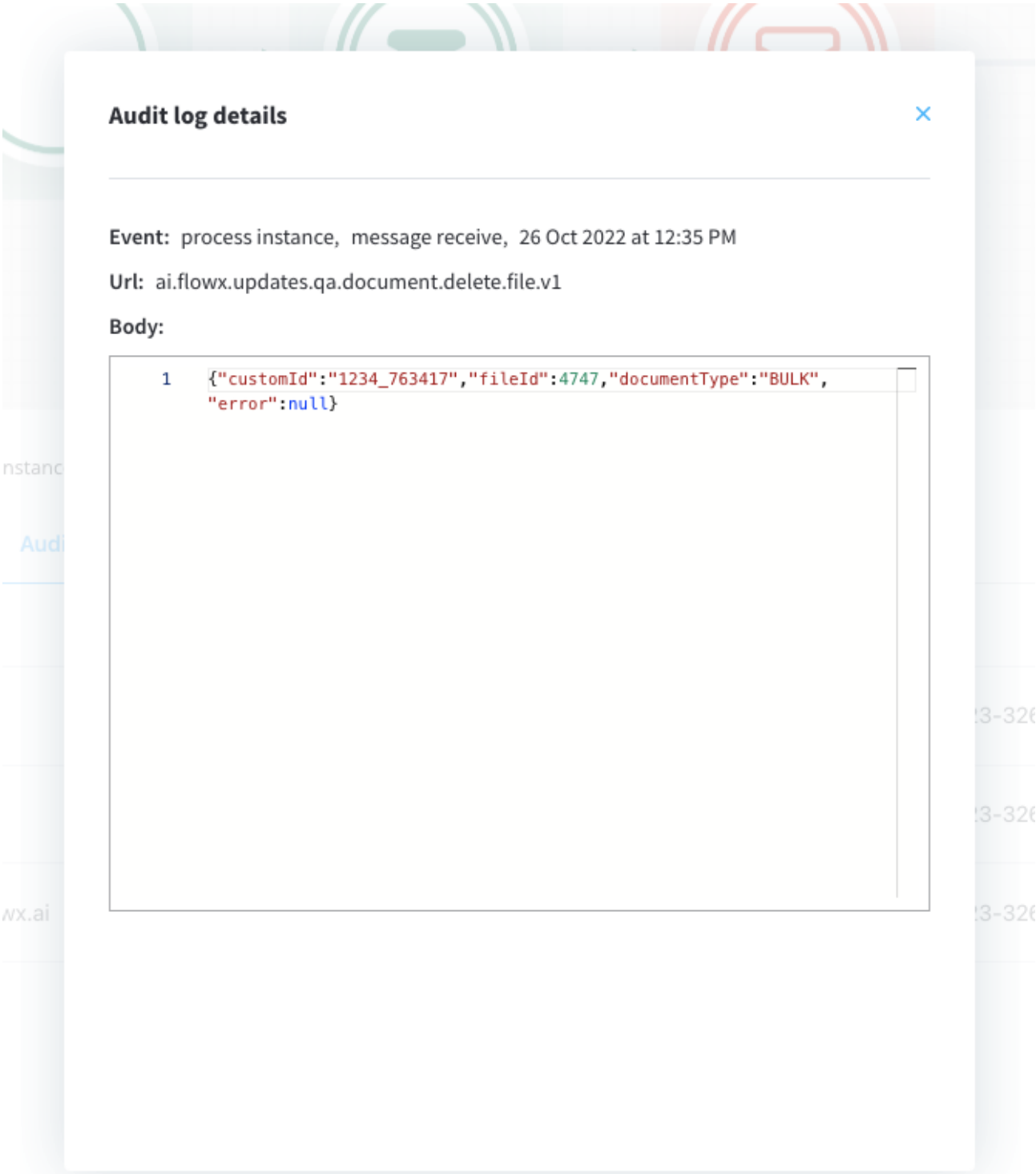
4. Fill in the body message request:

- `fileId` - the id of the file
- `customId` - the client ID
- `documentType` - document type

> **INFO**
>
> Kafka topic names can be set by using (overwriting) the following environment variables in the deployment:
>
> `KAFKA_TOPIC_FILE_DELETE_IN` - default value: `ai.flowx.in.qa.document.delete.file.v1`
>
> `KAFKA_TOPIC_FILE_DELETE_OUT` - default value: `ai.flowx.updates.document.delete.file.v1`

The Engine is listening for messages on topics with names of a certain pattern, make sure to use an outgoing topic name that matches the pattern configured in the Engine. :::

Receiving the reply



Values expected in the reply body:

- customId = client ID
- fileId = file ID
- documentType = document type
- error = error description

Example:

```
{
  "customId": "1234_763417",
  "fileId": 4747,
  "documentType": "BULK",
  "error": null
}
```

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Getting URLs for documents

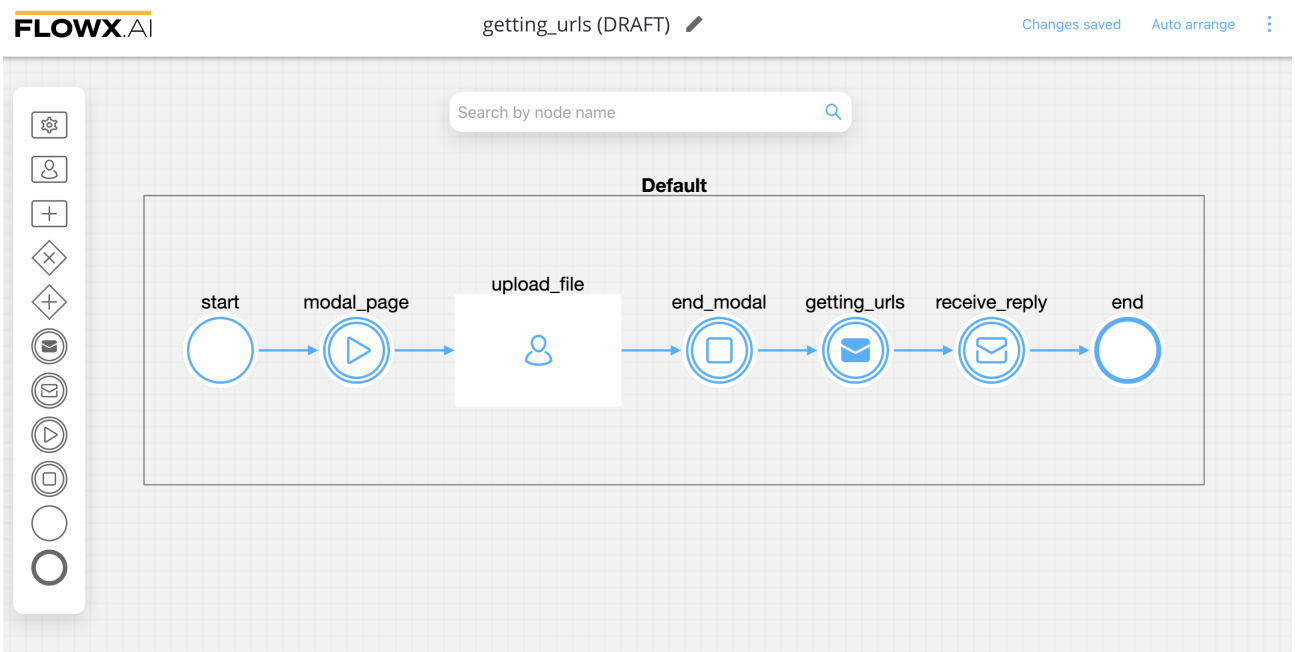
In certain scenarios, you may need to obtain URLs that point to uploaded documents to be used by other integrations. This requires adding a custom action to your process that requests the URLs from the Documents Plugin.

Sending the request

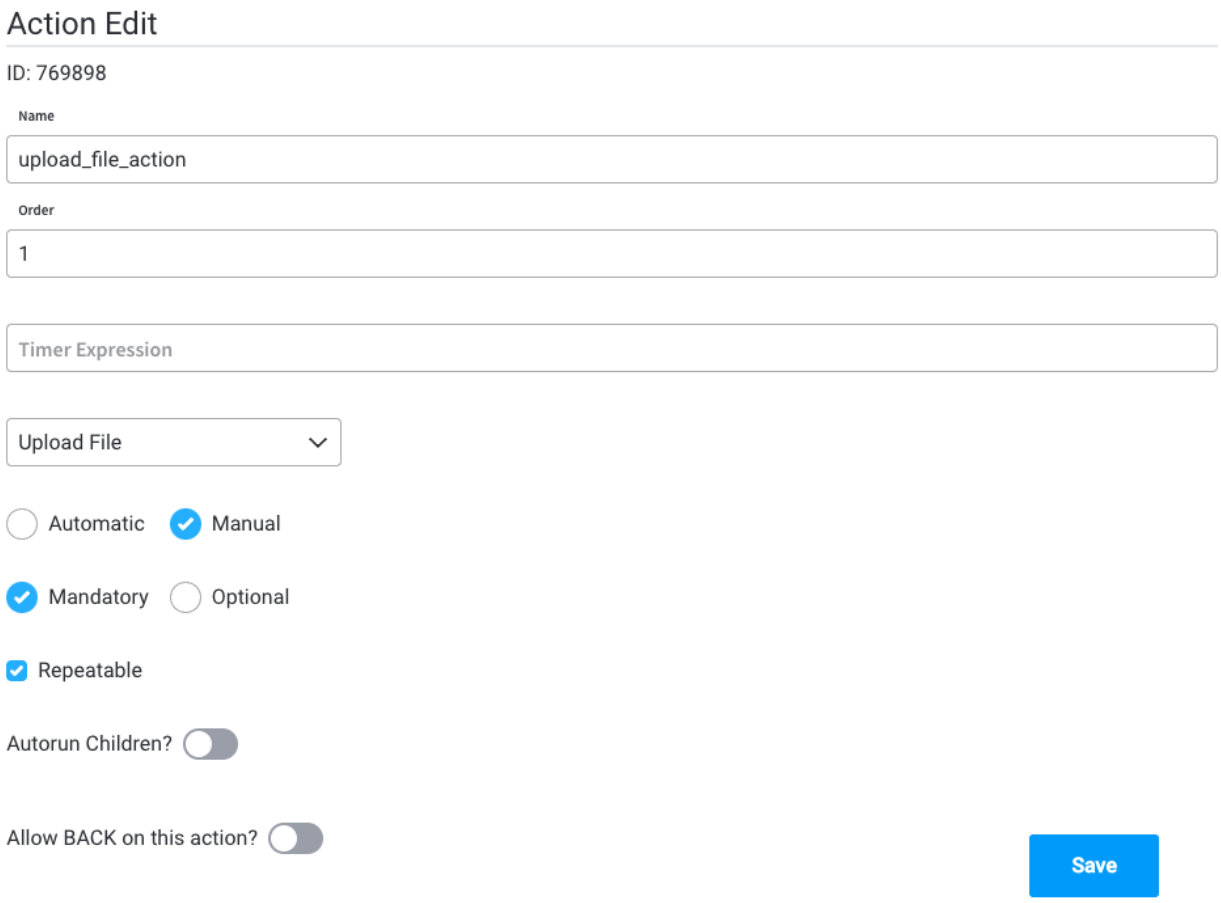
To retrieve document URLs and use them, for example, in the Notification Plugin to attach them to emails, follow the next steps:

1. Create a process and include the following nodes:

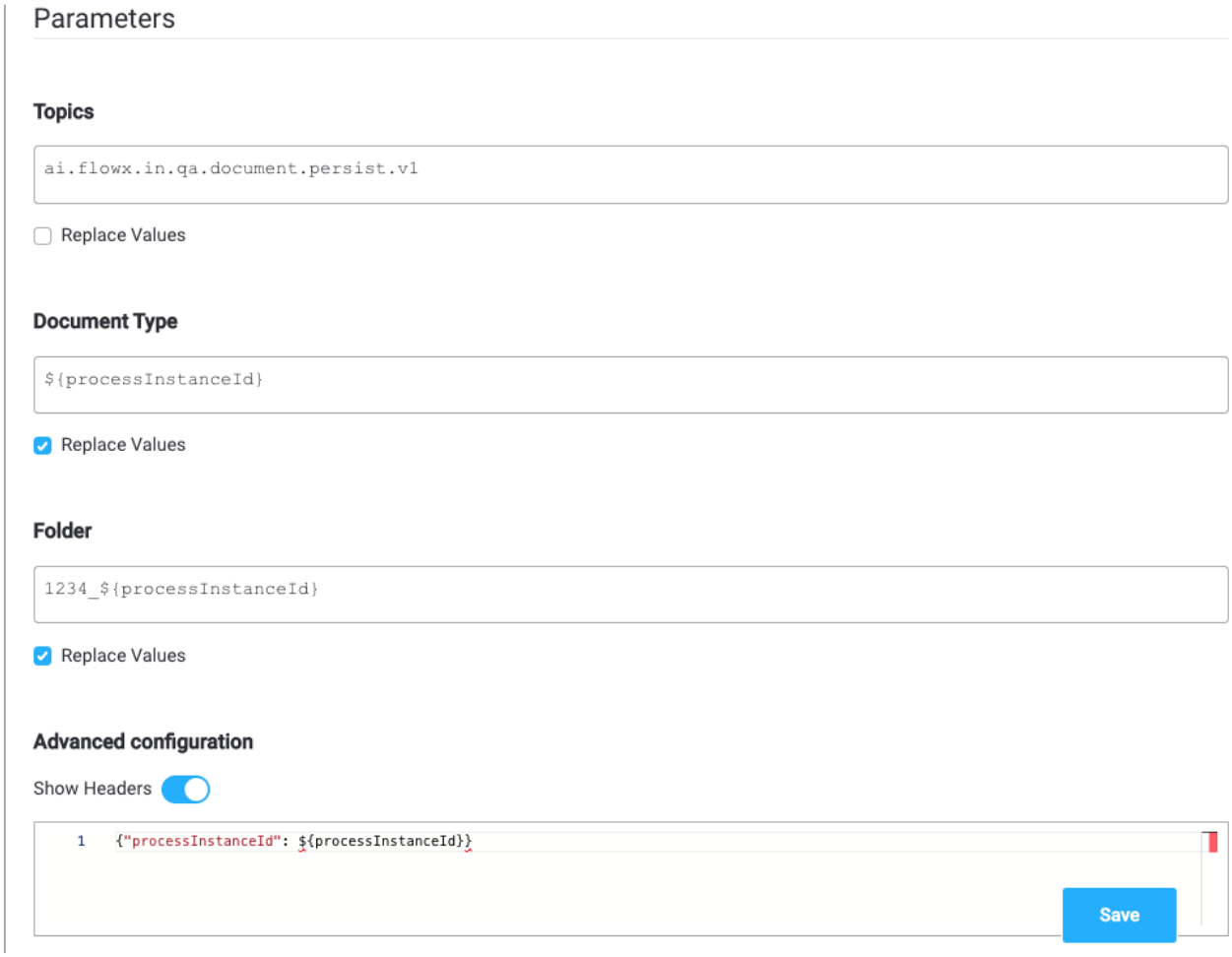
- a **Kafka Send Event Node**,
- a **Kafka Receive Event Node**
- a **User Task Node**
- **Start / End <ilestone** Nodes to create a modal



2. Configure the **User Task Node** and add an **Upload Action** to it.

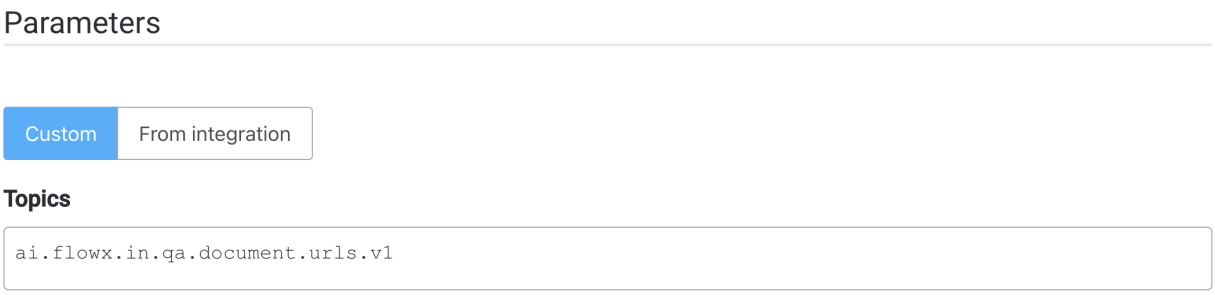


3. Configure the parameters for the **Upload Action**:

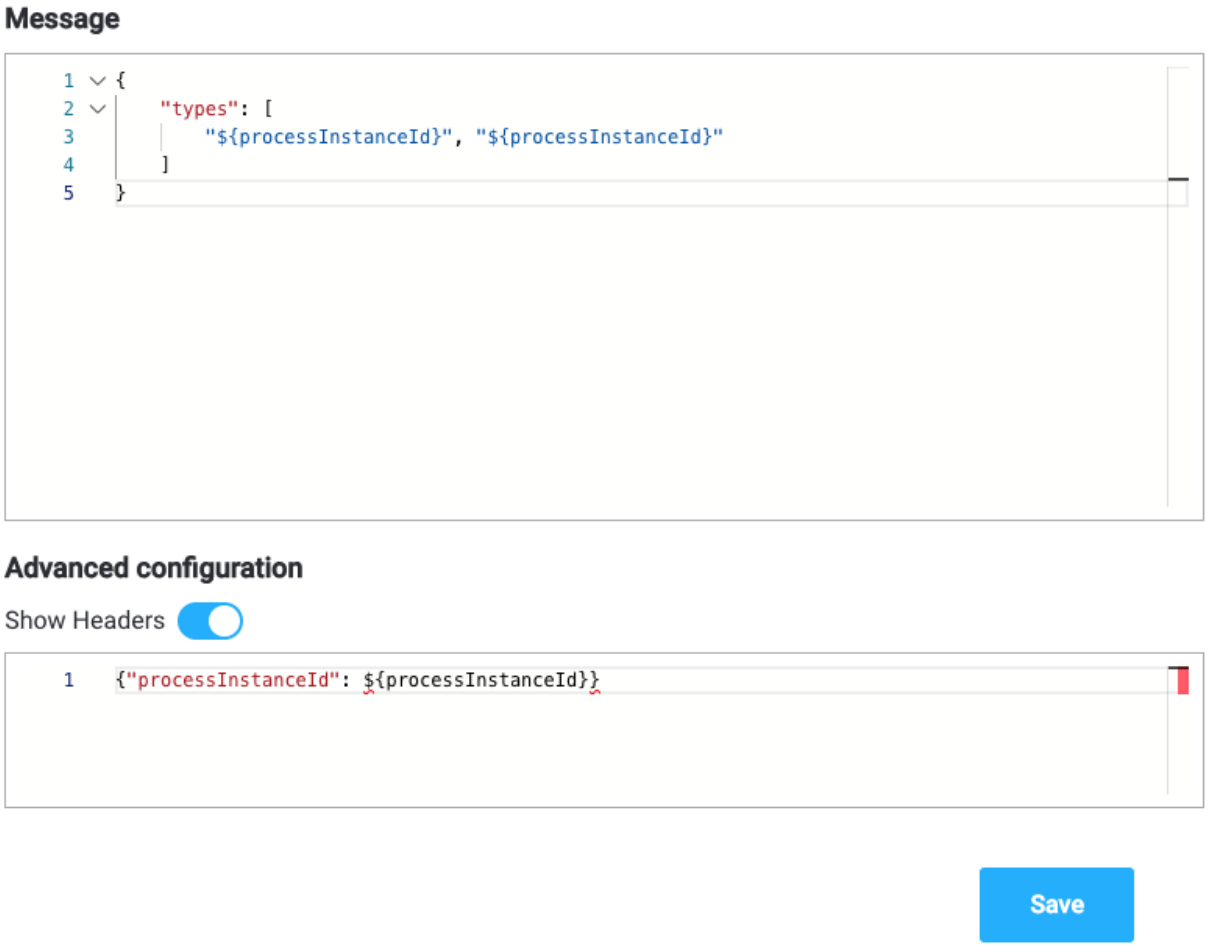


> **INFO**
>
> For more details on uploading a document and configuring an upload action, refer to the following sections:
>
> **Upload document**
>
> **Upload action**

4. Configure the Kafka Send Event Node by adding a **Kafka Send Action** and specifying the **Kafka topic** to send the request to:

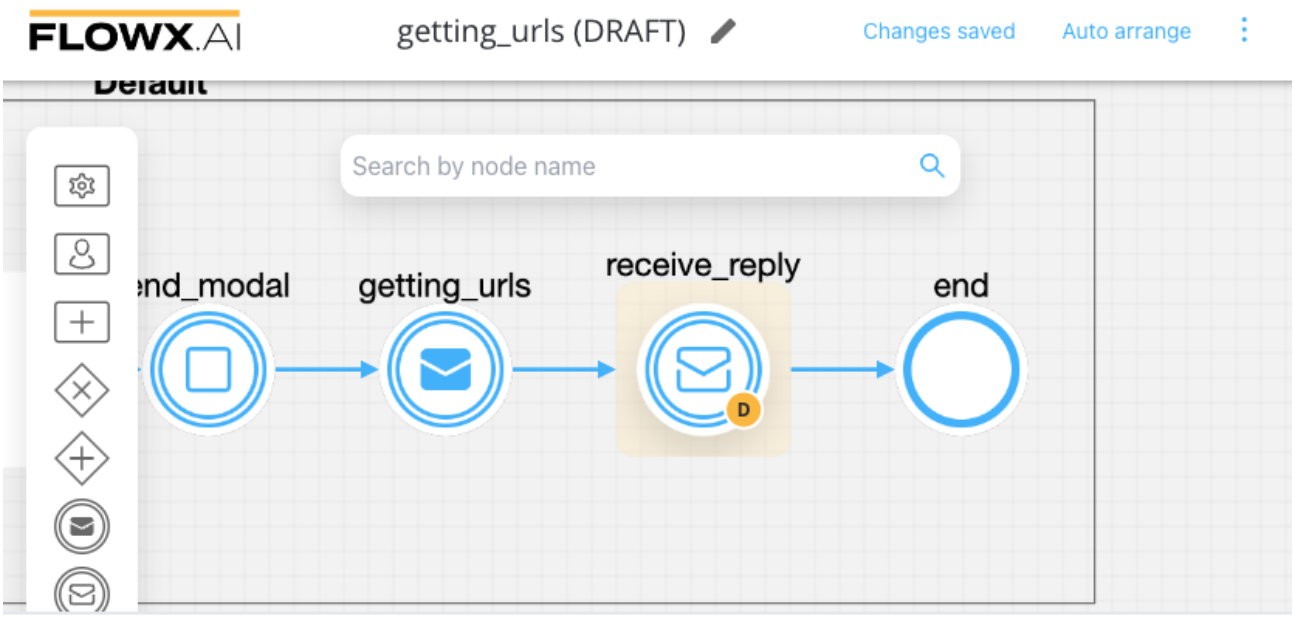


5. Fill in the body of the request message for the action:



- `types` - a list of document types

6. Configure the **Kafka Receive Event Node** by adding the kafka topic on which the response will be sent.



Node: **receive_reply** (ID: 766487)

Node Config

Swimlane

Default

Stage

Response Timeout

Response Timeout (PT30S)

Data stream topics

Custom | From integration

Topic Name: ai.flowx.updates.qa.documen

Key Name: receiveReply

Save

INFO

Kafka topic names can be set by using environment variables:

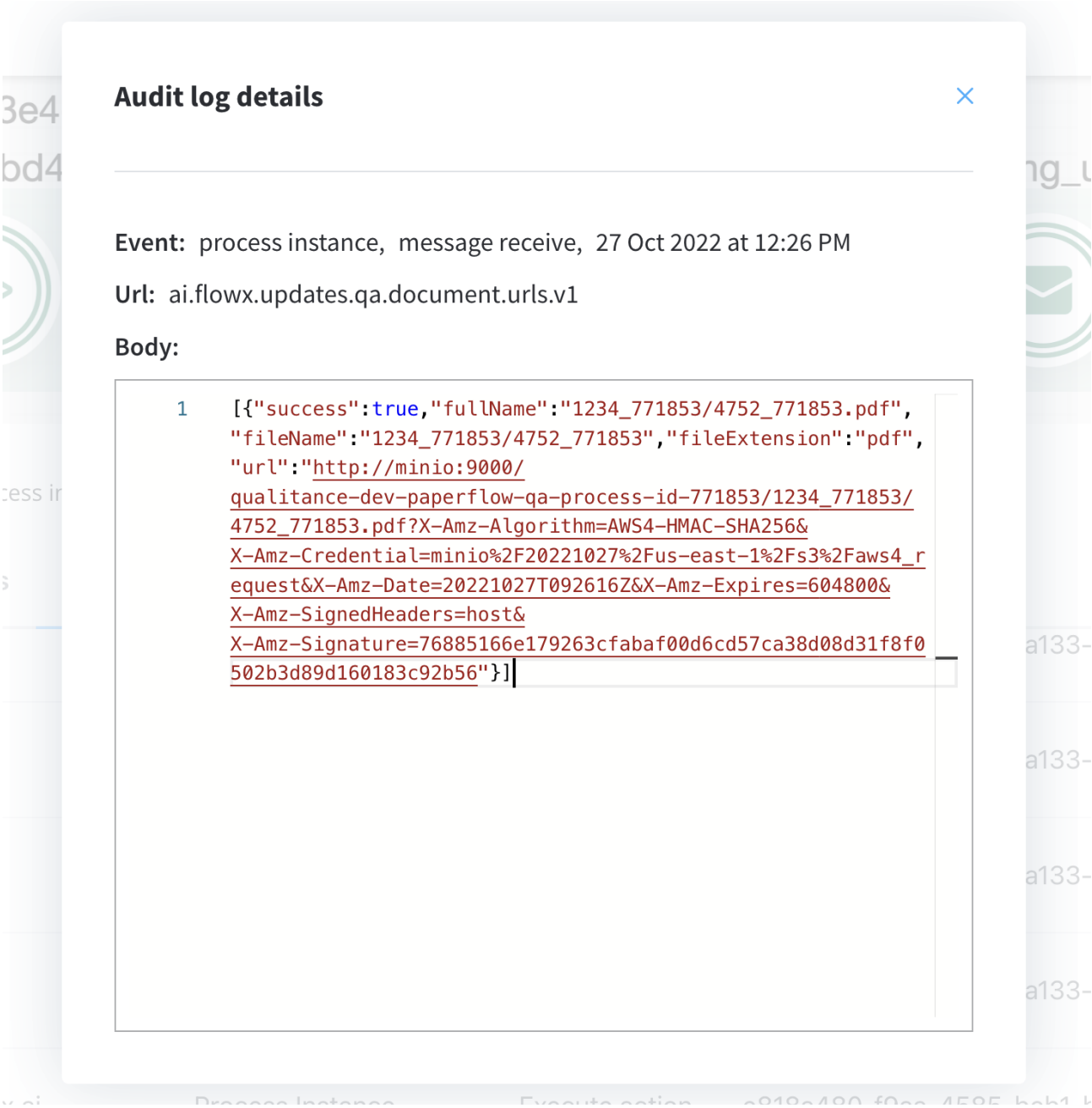
- `KAFKA_TOPIC_DOCUMENT_GET_URLS_IN` - `ai.flowx.in.qa.document.urls.v1` - the topic that listens for the request from the engine
- `KAFKA_TOPIC_DOCUMENT_GET_URLS_OUT` - `ai.flowx.updates.qa.document.urls.v1` - the topic on which the engine will expect the reply

The example topic names above are from an internal testing environment. When setting topics for other environments, follow this pattern: `ai.flowx.updates.{{environment}}.document.urls.v1`.

CAUTION

The Engine listens for messages on topics with specific naming patterns. Ensure that your outgoing topic name matches the pattern configured in the Engine.

Receiving the reply



The response body is expected to contain the following values:

```
[
    {
        "success": true,
```

```
        "fullName": "1234_771853/4752_771853.pdf",
        "fileName": "1234_771853",
        "fileExtension": "pdf",
        "url": "
<http://SOME_URL/1234_771853/4752_771853.pdf?X-Amz-
Algorithm=SOME_ALGORITHM&X-Amz-Credential=SOME_CREDENTIAL&X-
Amz-Date=20210223T113621Z&X-Amz-Expires=604800&X-Amz-
SignedHeaders=host&X-Amz-Signature=>"
    }
]
```

- **success**: A boolean indicating whether the document exists and the URL was generated successfully.
- **fullName**: The full name of the document file, including the directory path.
- **fileName**: The name of the document file without the extension.
- **fileExtension**: The extension of the document file.
- **url**: The full download URL for the document.

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Documents plugin / Using the plugin / Listing stored files

If you are using an S3-compatible cloud storage solution such as MinIO, the stored files are organized into buckets. A bucket serves as a container for objects stored

in Amazon S3. The Documents Plugin provides a REST API that allows you to easily view the files stored in the buckets.

To determine the partitioning strategy used for storing generated documents, you can access the following key in the configuration:

`application.file-storage.partition-strategy`

```
application:
  defaultLocale: en
  supportedLocales: en, ro
  jaeger.prefix: document
  #fileStorageType is the configuration that activates one
FileContentService implementation. Valid values: minio /
fileSystem
  file-storage:
    type: s3
    disk-directory: MS_SVC_DOCUMENT
    partition-strategy: NONE
```

The `partition-strategy` property can have two possible values:

- **NONE**: In this case, documents are saved in separate buckets for each process instance, following the previous method. **PROCESS_DATE**: Documents are saved in a single bucket with a subfolder structure based on the process date. For example: `bucket/2022/2022-07-04/process-id-xxxx/customer-id/file.pdf`.

REST API

The Documents Plugin provides the following REST API endpoints for interacting with the stored files:

List buckets

GET documentURL/internal/storage/buckets

This endpoint returns a list of available buckets.

List Objects in a Bucket

GET documentURL/internal/storage/buckets/BUCKET_NAME

This endpoint retrieves a list of objects stored within a specific bucket. Replace `BUCKET_NAME` with the name of the desired bucket.

Download File

GET documentURL/internal/storage/download

This endpoint allows you to download a file by specifying its path or key.

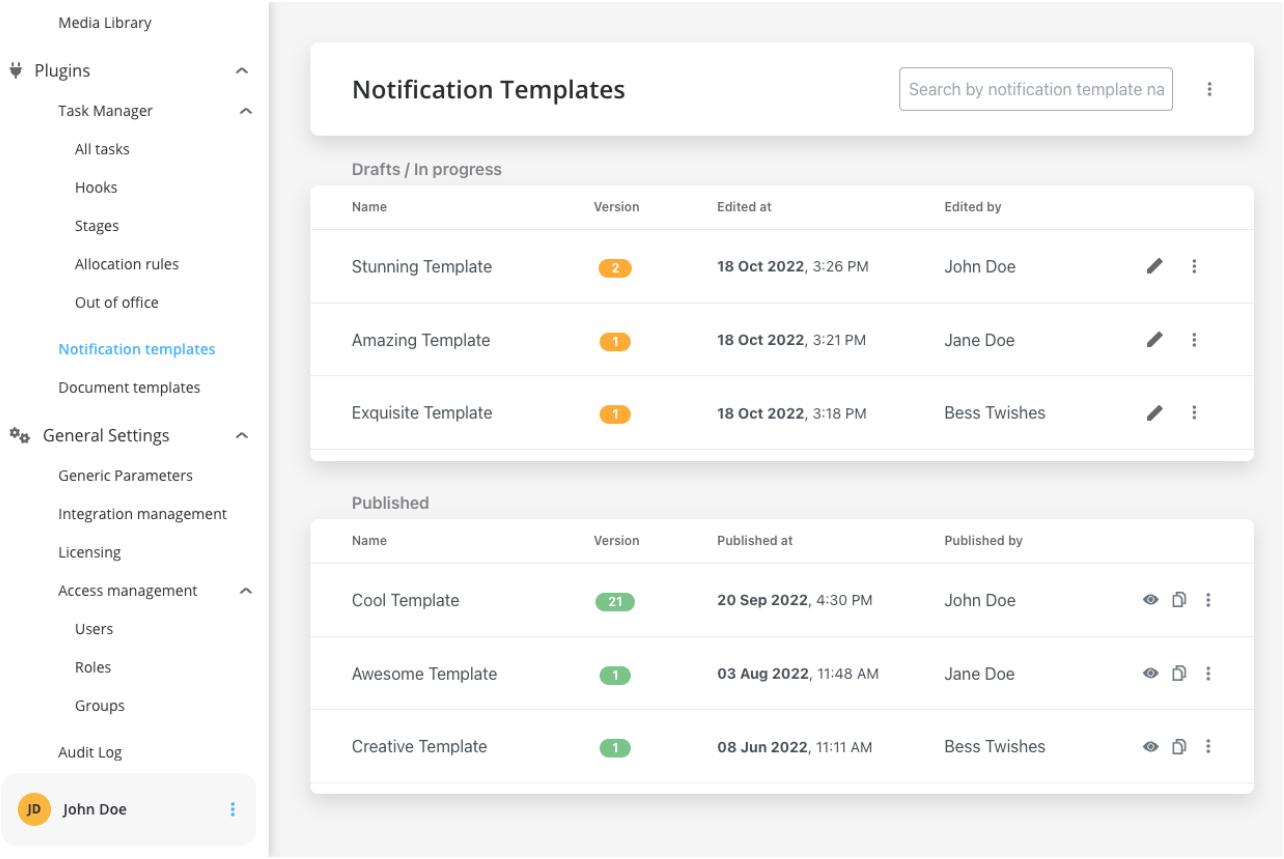
Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Notifications

plugin / Using the plugin / Managing notification templates

You can create and manage notification templates using

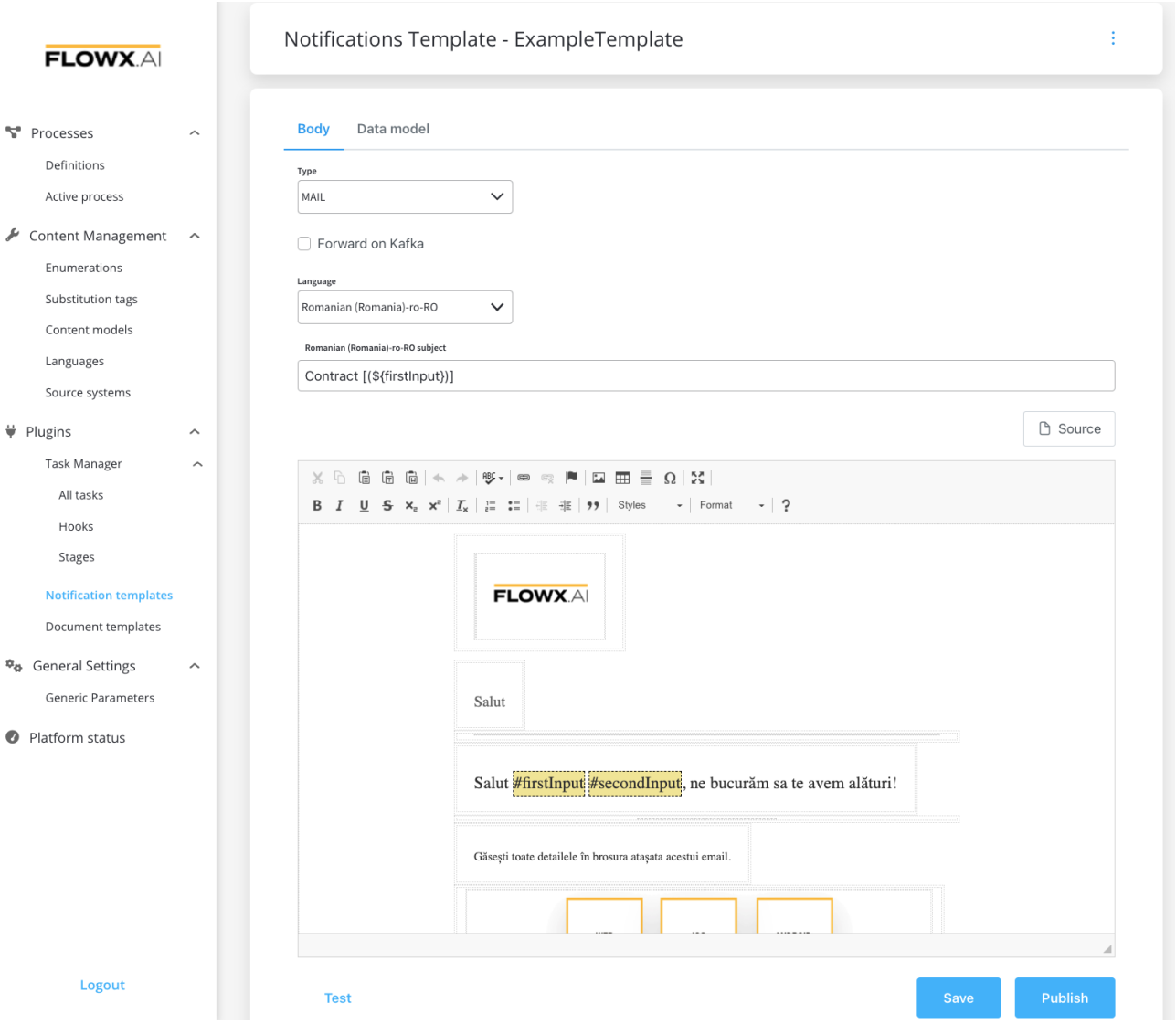
The fallback content to display on prerendering
web app, by accessing the dedicated section.



Configuring a template

To configure a document template, first, you need to select some information stored in the **Body**:

1. **Type** - could be either MAIL or SMS notifications
2. **Forward on Kafka** - if this checkbox is ticked, the notification is not being sent directly by the plugin to the destination, but forwarded to another adapter (this is mandatory for SMS notifications templates, as they require an external adapter)
3. **Language** - choose the language for your notification template
4. **Subject** - enter a subject



Editing the content

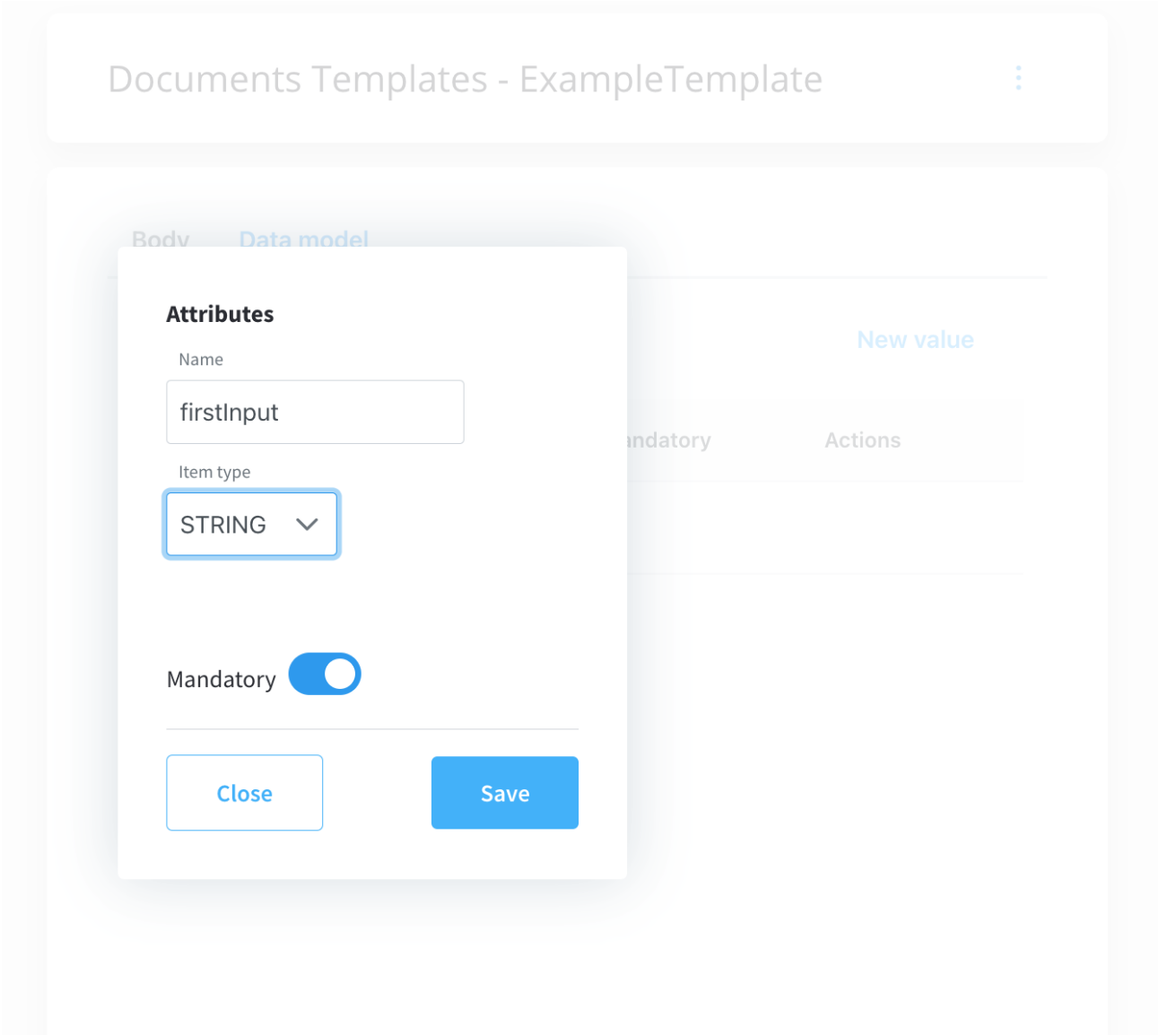
You can edit the content of a notification template by using the WYSIWYG editor embedded in the body of the notification templates body.

Configuring the data model

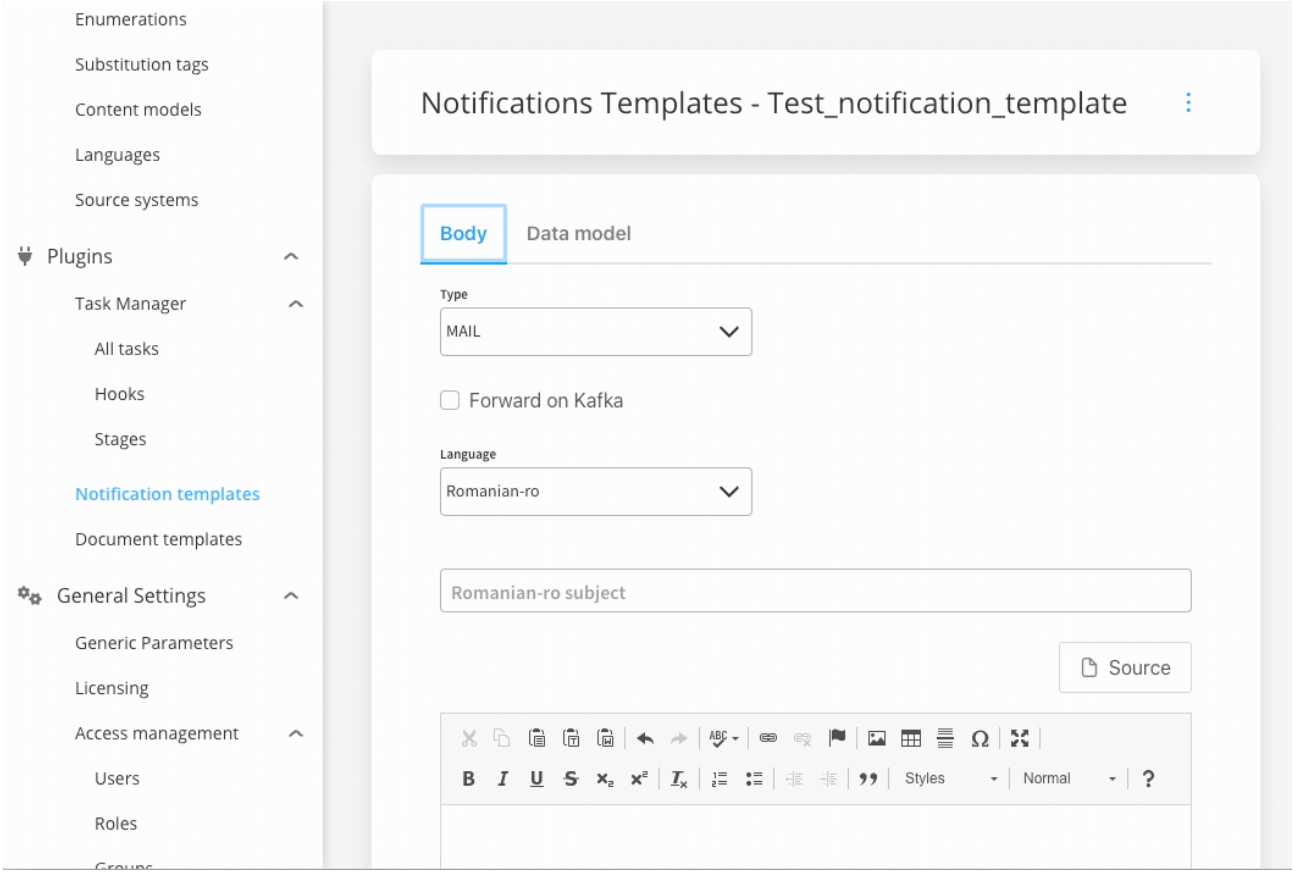
Using the

The fallback content to display on prerendering
, you can define key pair values (parameters) that will be displayed and reused in the editor. Multiple parameters can be added:

- STRING
- NUMBER
- BOOLEAN
- OBJECT
- ARRAY (which has an additional `item` field)



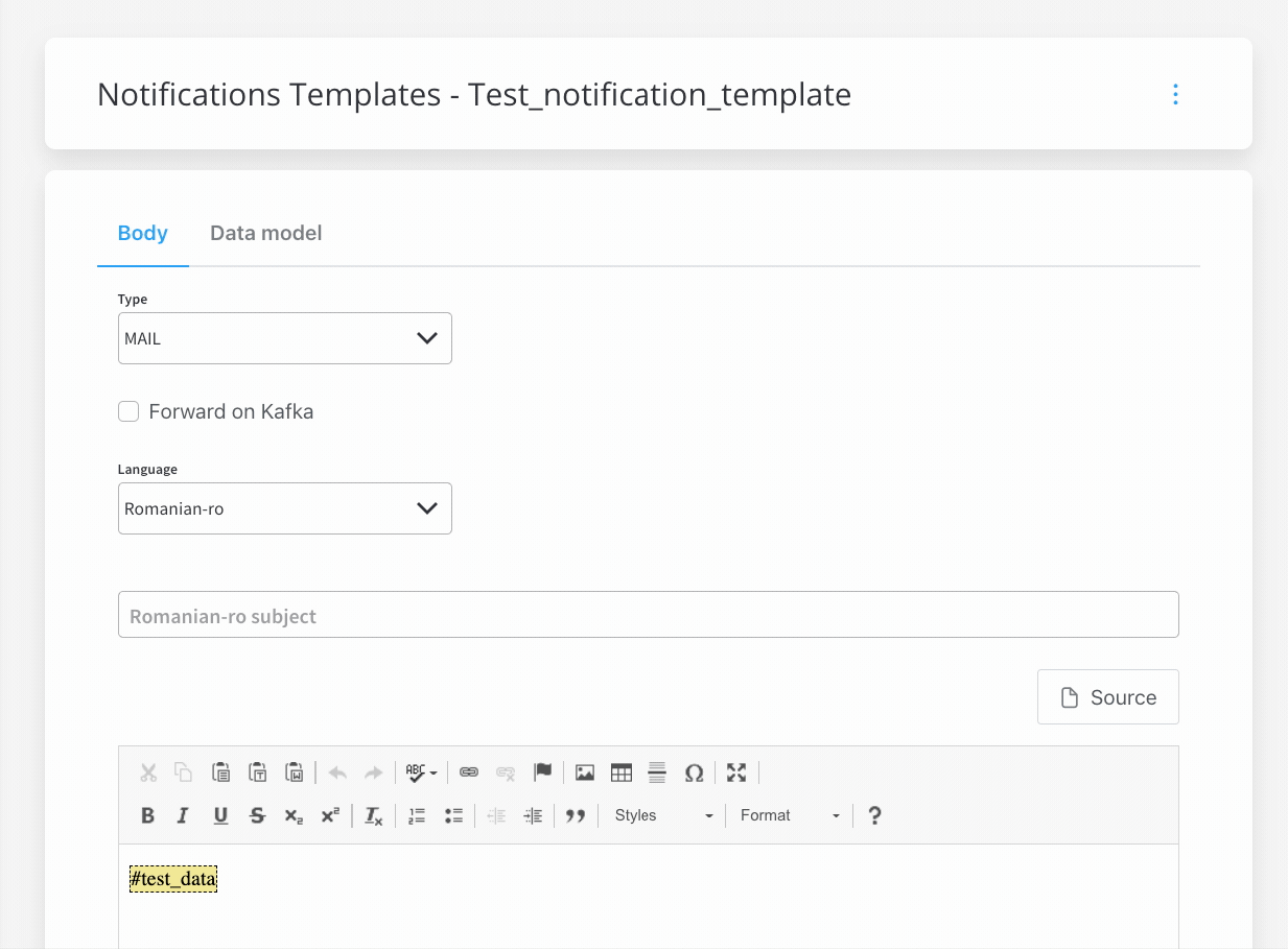
After you defined some parameters in the **Data Model** tab, you can type "**#**" in the editor to trigger a dropdown where you can choose which one you want to use/reuse.



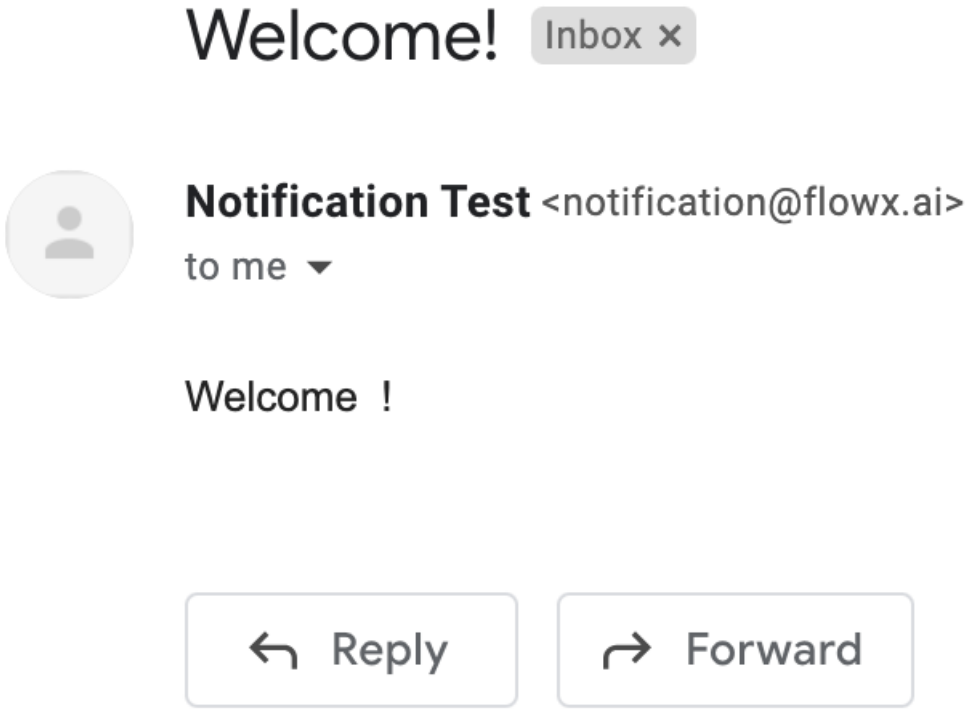
» WYSIWYG Editor

Testing the template

You can use the test function to ensure that your template configuration is working as it should before publishing it.



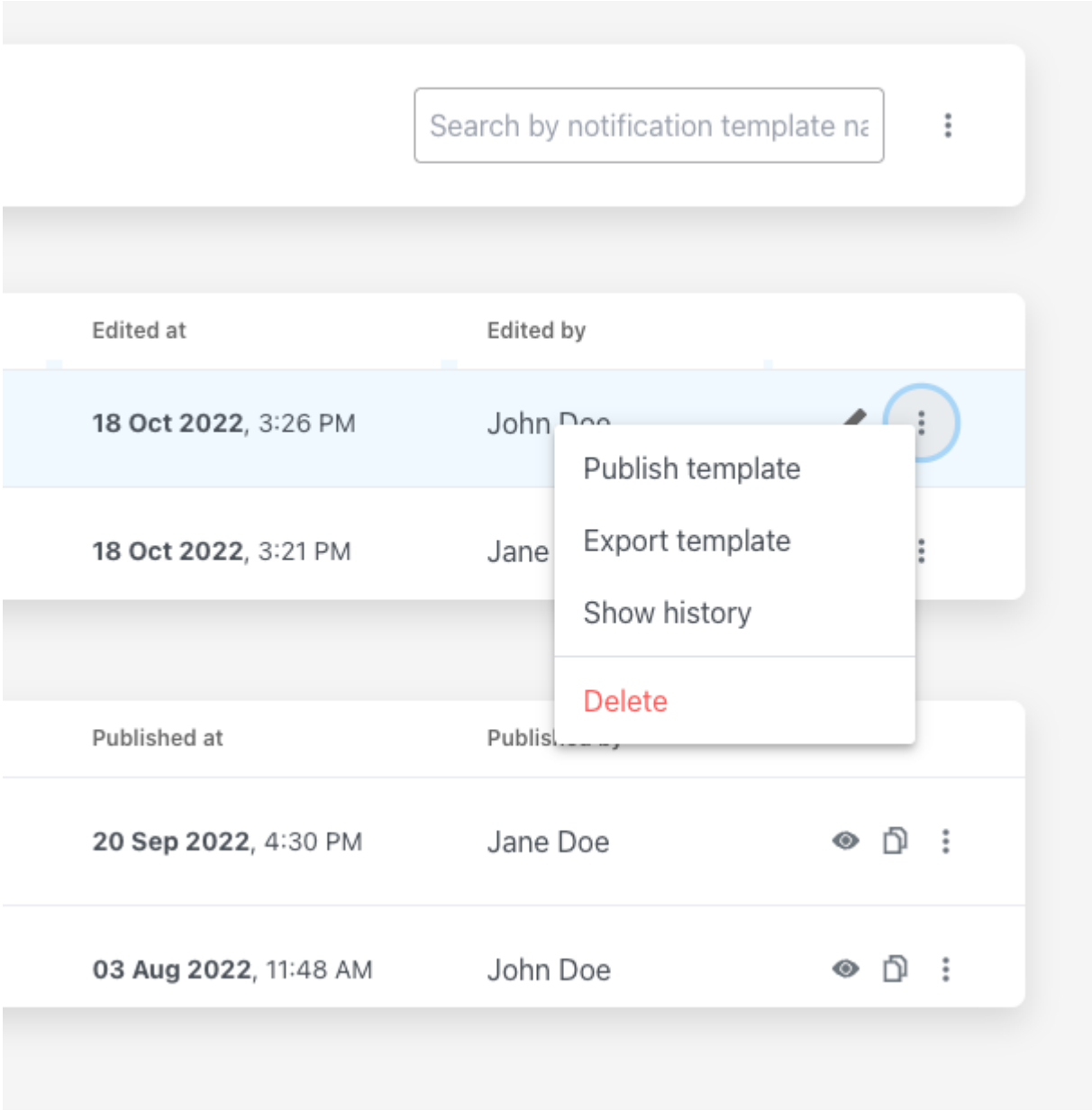
In the example above, some keys (marked as mandatory) were not used in the template, letting you know that you've missed some important information. After you enter all the mandatory keys, the notification test will go through:



Other actions

When opening the contextual menu (accessible by clicking on the breadcrumbs button), you have multiple actions to work with the notifications templates:

- Publish template - publish a template (it will be then displayed in the **Published** tab), you can also clone published templates
- Export template - export a template (JSON format)
- Show history - (version history and last edited)



Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Notifications

plugin / Using the plugin / Sending a notification

The plugin can be used for sending many kinds of notifications such as emails or SMS notifications. It can be easily integrated in one of your business processes.

Configuring the process

To configure a business process that sends notifications you must follow the next steps:

- use
  The fallback content to display on prerendering
  web app to create/edit a notification template
- use
  The fallback content to display on prerendering
  to add a **Message send task** and a **Message received task**
- configure the needed actions
- configure the request body
- configure the needed Kafka topics

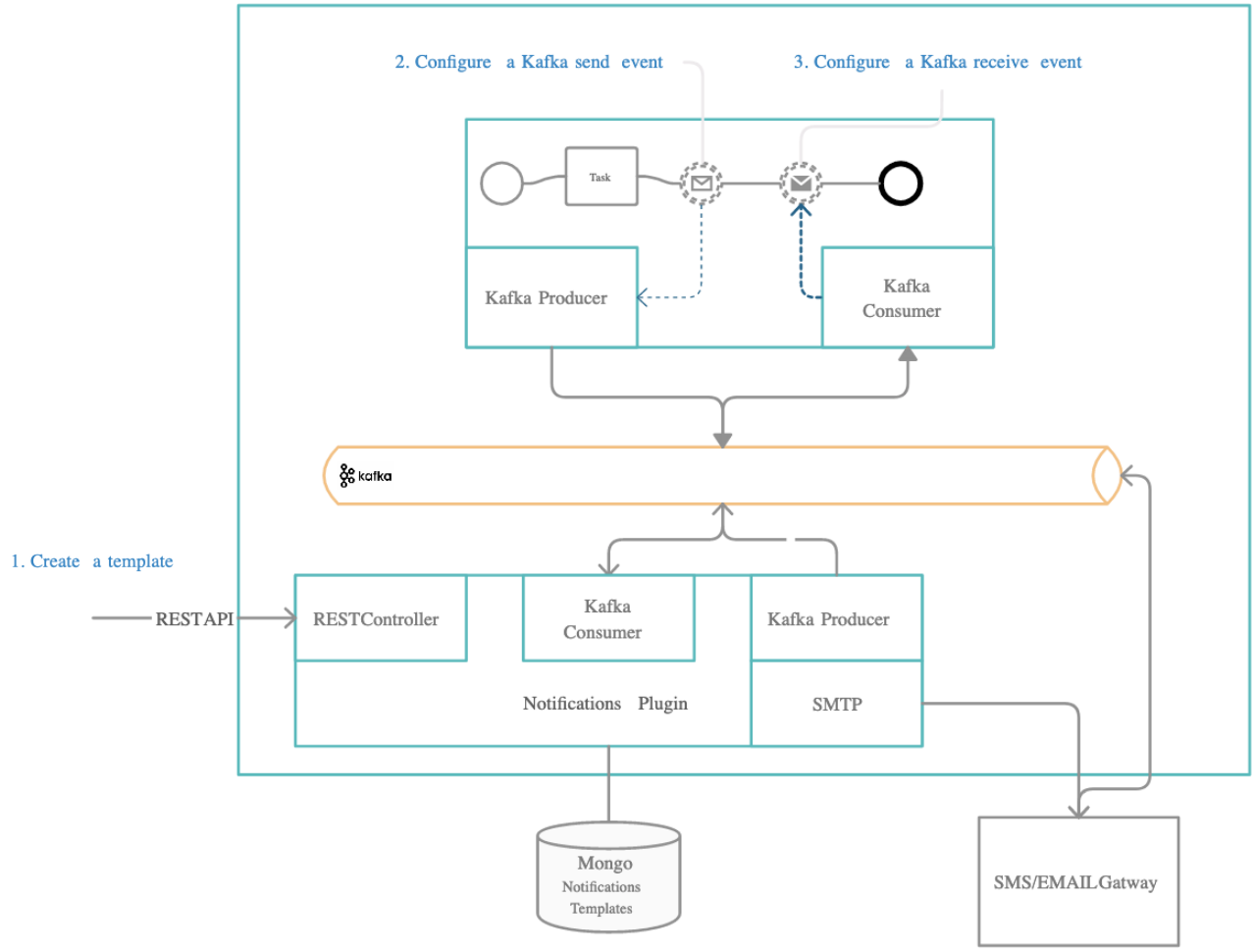
The following values are expected in the request body:

Key	Definition	
language	The language that should be used	Mandatory

Key	Definition	
templateName	The name of the notification template that is used	Mandatory
channel	Notification channel: SMS/MAIL	Mandatory
receivers	Notification receivers: email/phone number	Mandatory
senderEmail	Notification sender email	Optional
senderName	Notification sender name	Optional
attachments	Attachments that are sent with the notification template (only used for MAIL notifications)	Optional

INFO

Check the detailed example below.

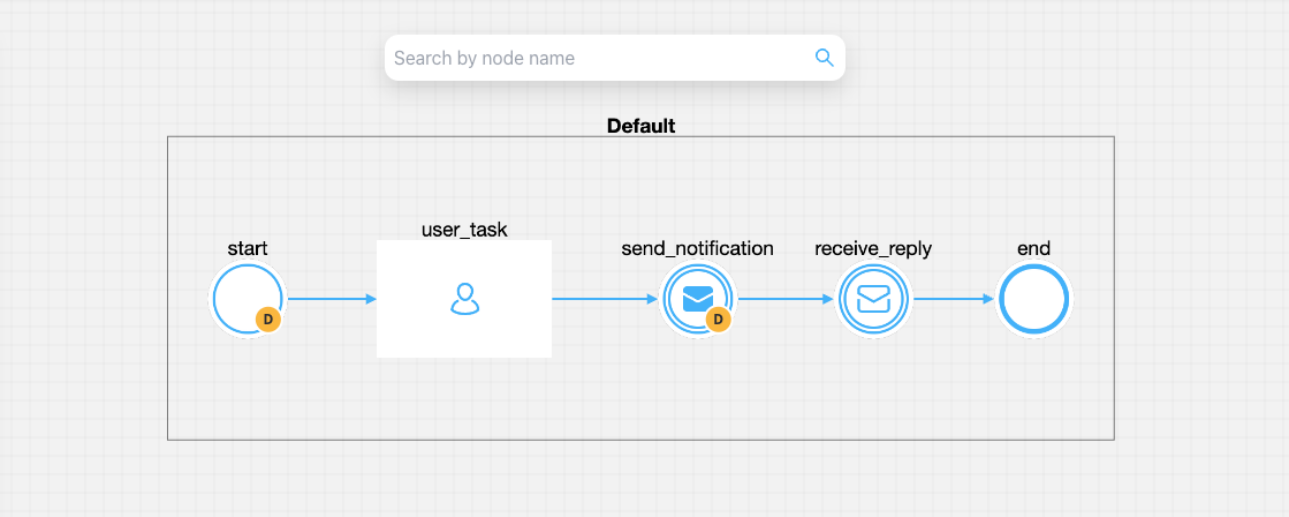


Define needed Kafka topics

Kafka topic names can be set by using environment variables:

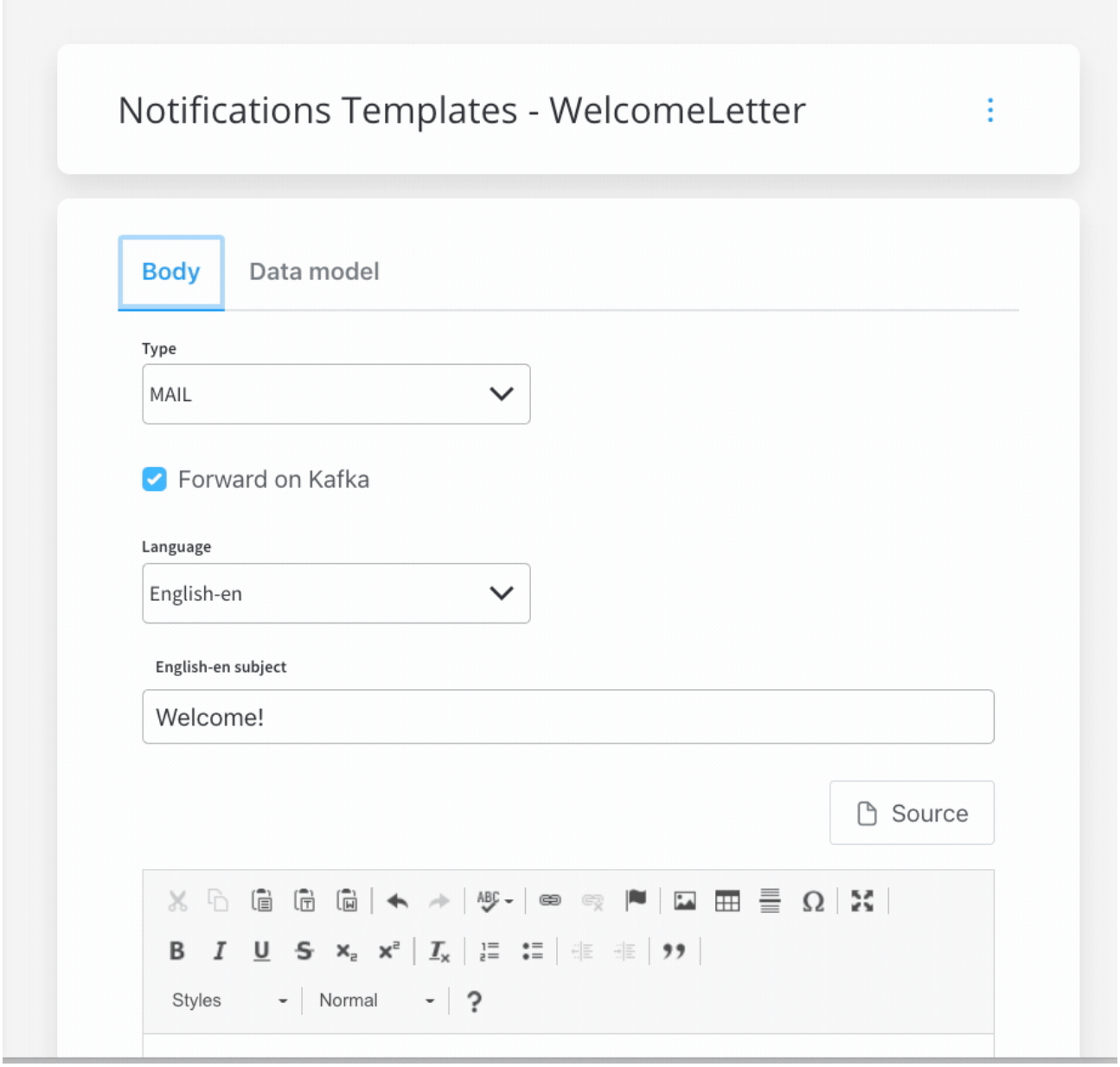
- `KAFKA_TOPIC_NOTIFICATION_INTERNAL_IN` - topic used to trigger the request to send a notification
- `KAFKA_TOPIC_NOTIFICATION_INTERNAL_OUT` - topic used for sending replies after sending the notification

Example: send a notification from a business flow



Let's pick a simple use-case, say we need to send a new welcome letter when we onboard a new customer. The steps are the following:

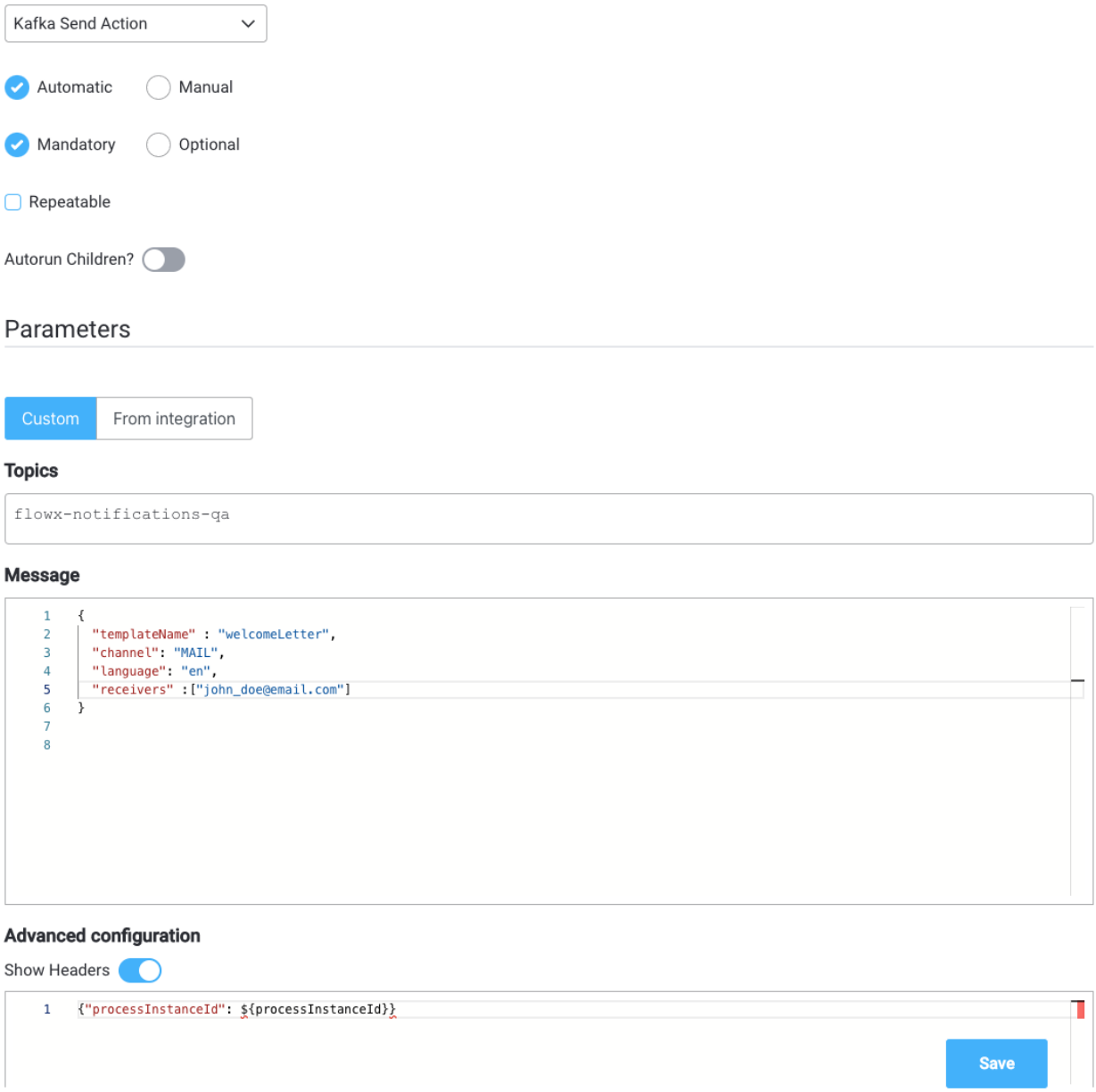
1. Configure the template that you want to use for the welcome email, see the previous section, Managing notification templates for more information.



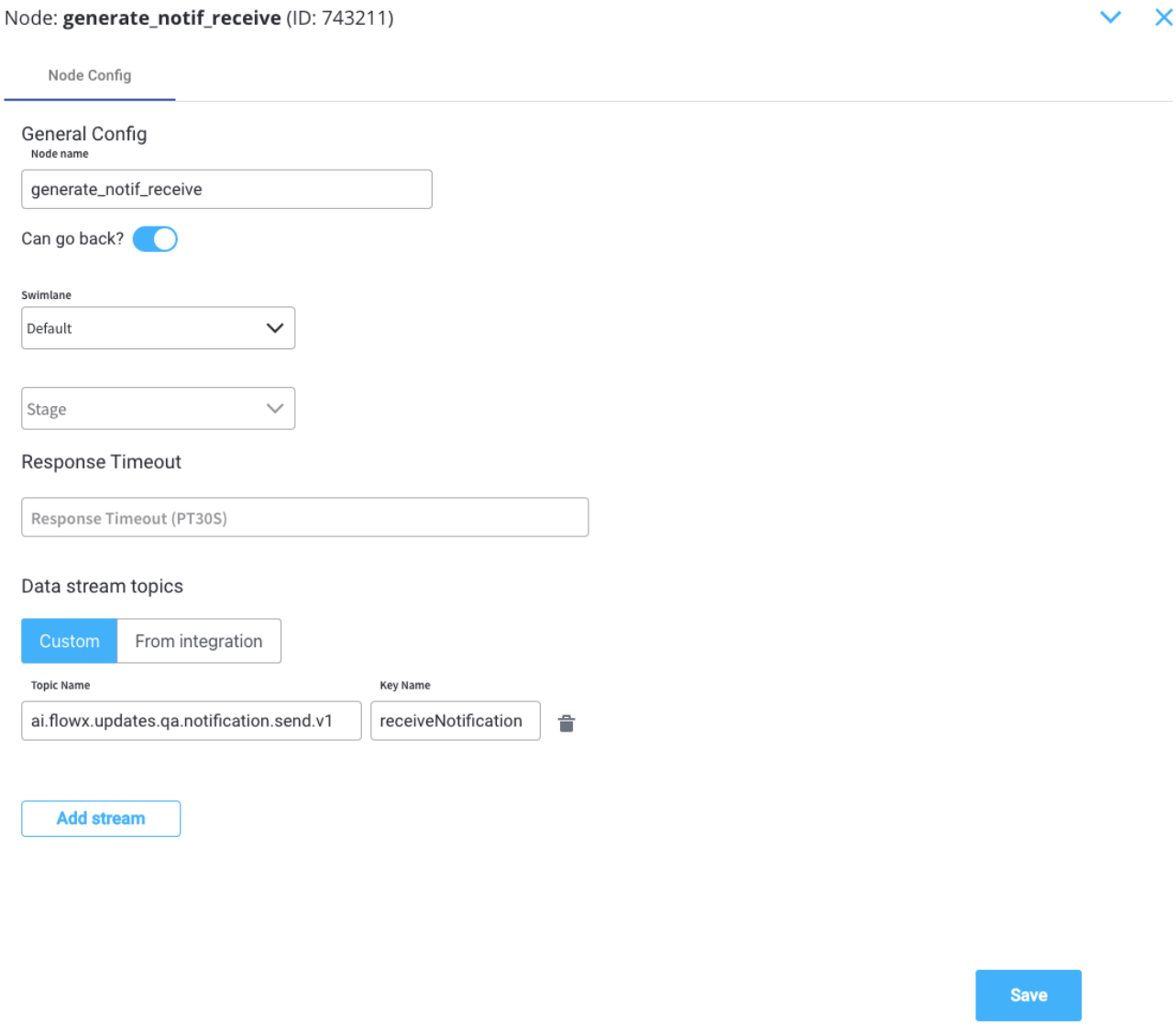
2. Use the FLOWX.AI Designer to add a **Message send task** and a **Message received task**.
3. On the **Message send task** add a proper configuration to the action, the Kafka topic and request body message to be sent:

- **Topics** - `KAFKA_TOPIC_NOTIFICATION_INTERNAL_IN` - `flowx-notifications-qa`

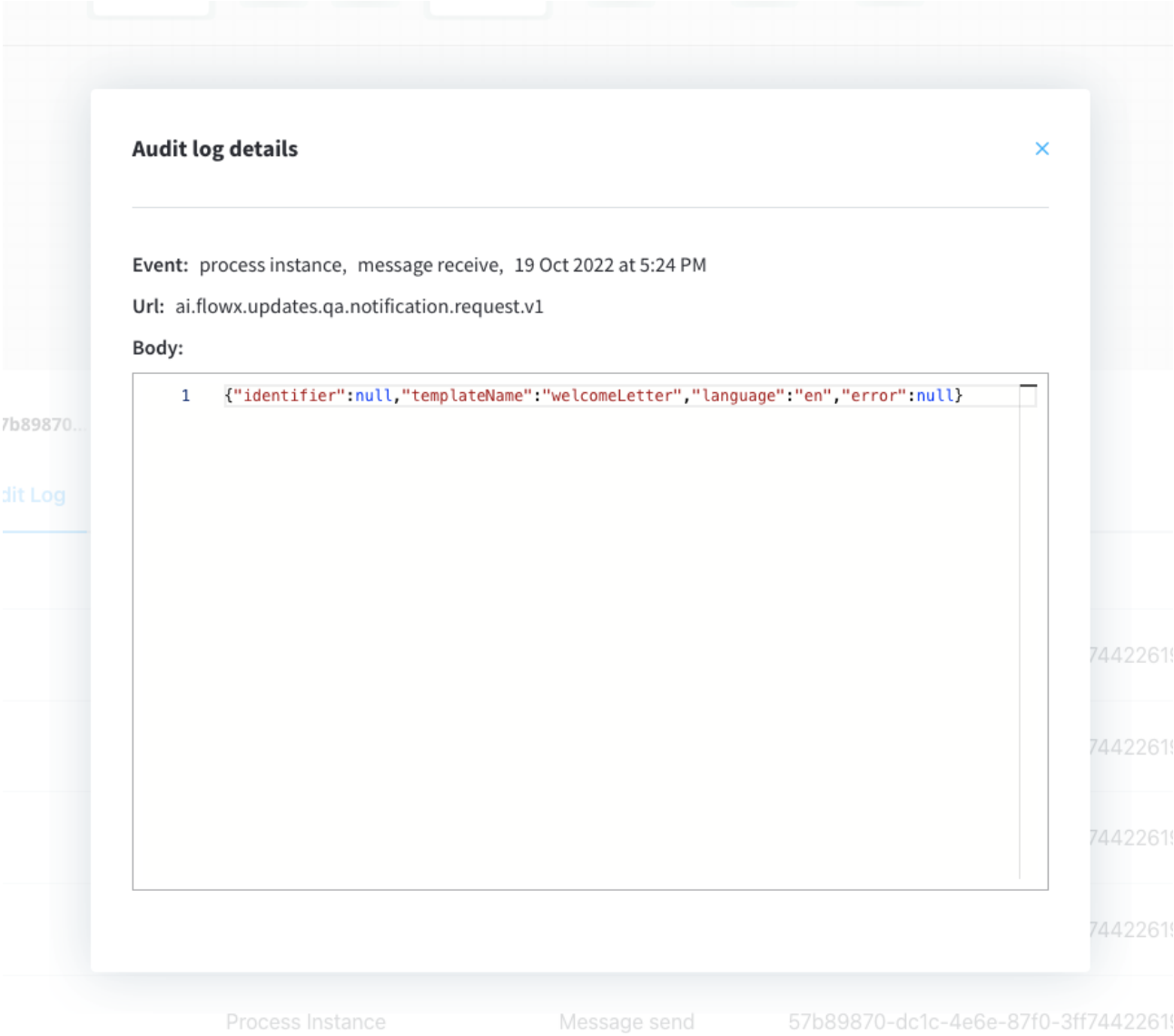
- **Message** (expected parameters):
  - templateName
  - channel
  - language
  - receivers
- **Headers** - it is always `{"processInstanceId": ${processInstanceId}}`



4. On the **Message received task** add the needed topic to receive the kafka response - `KAFKA_TOPIC_NOTIFICATION_INTERNAL_OUT` - `ai.flowx.updates.qa.notification.request.v1`.



5. Run the process and look for the response (you can view it via the **Audit log**) or checking the responses on the Kafka topic defined at `KAFKA_TOPIC_NOTIFICATION_INTERNAL_OUT` variable.



Response example at `KAFKA_TOPIC_NOTIFICATION_INTERNAL_OUT`:

```
{
  "identifier": null,
  "templateName": "welcomeLetter",
  "language": "en",
  "error": null
}
```

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Notifications plugin / Using the plugin / Sending an email with attachments

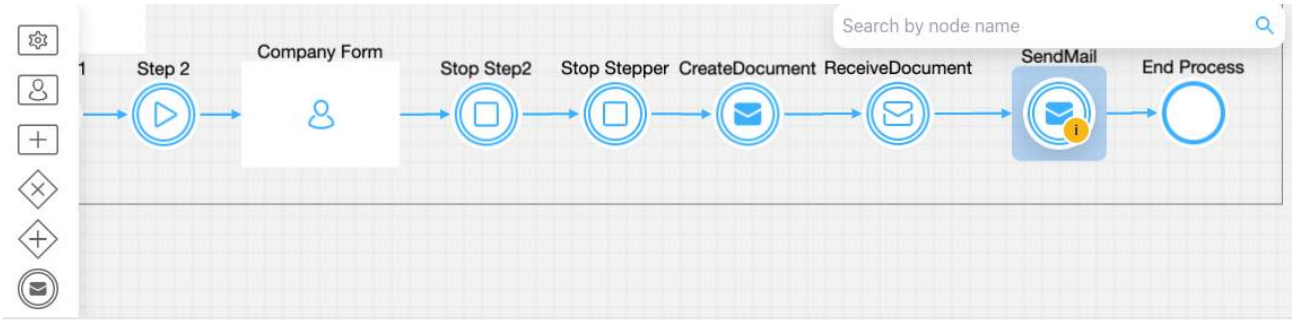
To use the notification plugin for sending emails with attachments, you must define the same topic configuration as for sending regular notifications. A notification template must be created, and the corresponding Kafka topics must be defined.

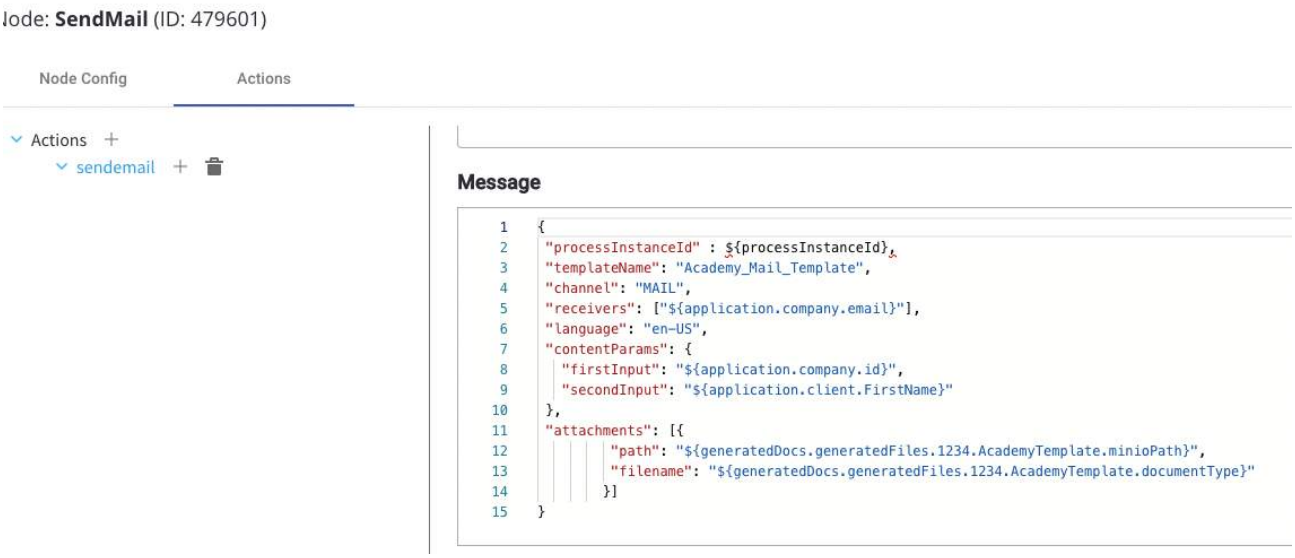
» Send a notification

Defining process actions

Example: send an email notification with attached files from a business flow

Let's pick a simple use-case. Imagine we need to send a copy of a contract signed by a new customer. Before setting the action for the notification, another action must be defined, so the first one will save the new contract using the documents plugin.





» Uploading a new document

The steps for sending the notification are the following:

Step 1: Configure the template that you want to use for the email, see the Managing notification templates section for more information.

Step 2: Check that the needed topics are defined correctly on the following environment variables:

`KAFKA_TOPIC_NOTIFICATION_INTERNAL_IN`

`KAFKA_TOPIC_NOTIFICATION_INTERNAL_OUT`

Step 3: Use the

The fallback content to display on prerendering
to add a new Kafka send event action to the correct node in the process definition.

Step 4: Add the proper configuration to the action, the Kafka topic and message to be sent.

The message to be sent to Kafka will look something like:

```
{
  "templateName" : "contractCopy",
  "identifier" : "text",
  "language": "en",
  "receivers" : [ "someone@somewhere.com" ],
  "contentParams" : {
    "clientId" : "clientId",
    "firstName" : "first",
    "lastName" : "last"
  },
  "attachments" : [ {
    "filename" : "contract",
    "path" : "MINIO_BUCKET_PATH/contract.pdf"
  } ]
}
```

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Notifications plugin / Using the plugin / Forward notifications to an external system

If the Notification service is not directly connected to an SMTP / SMS server and you want to use an external system for sending the notifications, you can use the notification plugin just to forward the notifications to your custom implementation.

Define needed Kafka topics

Kafka topic names can be set by using environment variables:

- `KAFKA_TOPIC_NOTIFICATION_INTERNAL_IN` - topic used to trigger the request to send a notification
- `KAFKA_TOPIC_NOTIFICATION_EXTERNAL_OUT` - the notification will be forwarded on this topic to be handled by an external system
- `KAFKA_TOPIC_NOTIFICATION_INTERNAL_OUT` - topic used for sending replies after sending the notification

Example: send a notification from a business flow

Let's pick a simple use case. Imagine we need to send a new welcome letter when we onboard a new customer. You must follow the next steps:

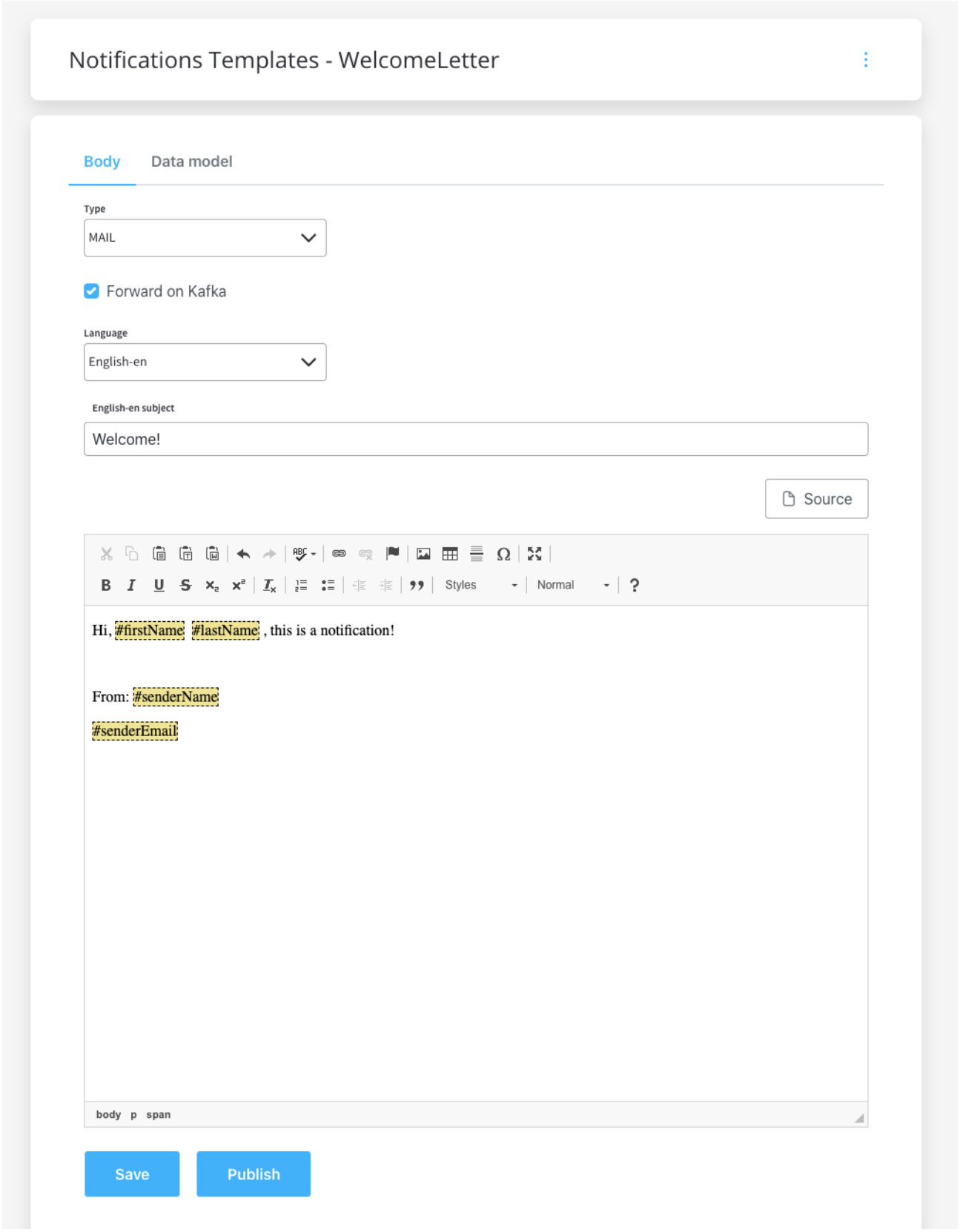
1. Configure the template that you want to use for the welcome email, use the WYSIWYG Editor

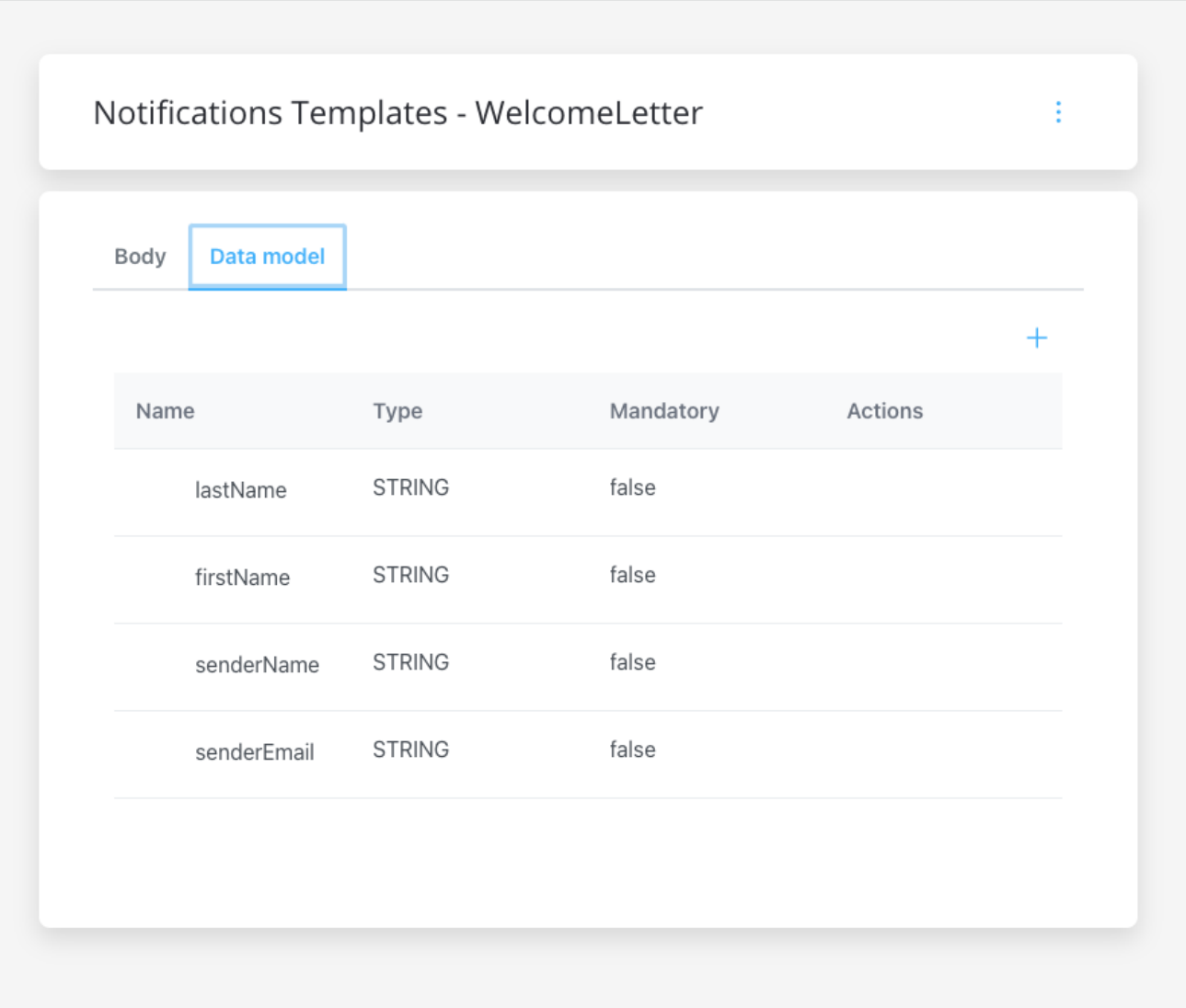
> ⚠ **CAUTION**
>
> Make sure that the **Forward on Kafka** checkbox is ticked, so the notification will be forwarded to an external adapter.

2. Configure the data model for the template.

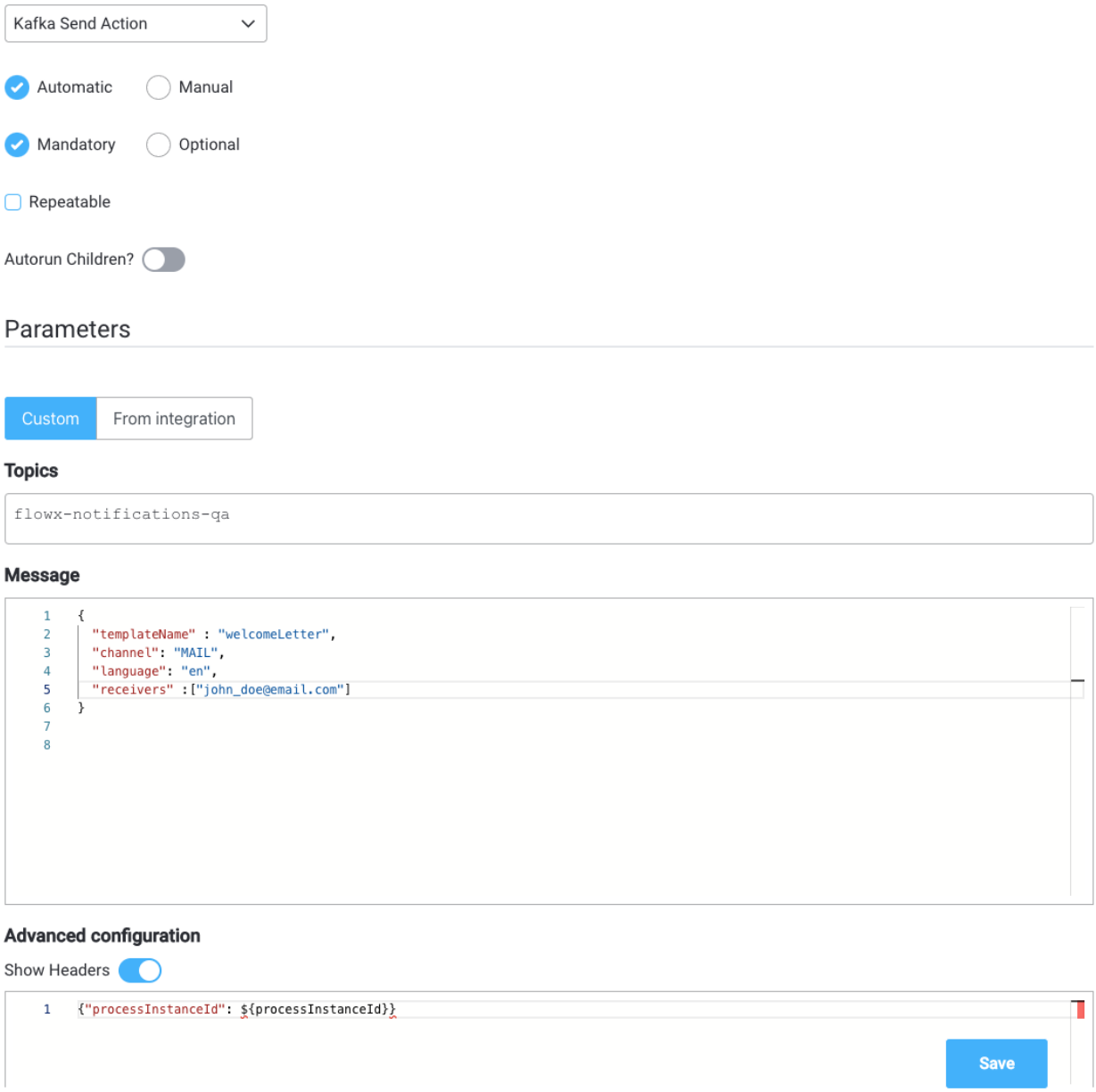
3. To configure a document template, first, you need to define some information stored in the Body:

- **Type** - MAIL (for email notifications)
- ❗ **Forward on Kafka** - if this box is checked, the notification is not being sent directly by the plugin to the destination, but forwarded to another adapter
- **Language** - choose the language for your notification template
- **Subject** - enter a subject


Notifications Templates - WelcomeLetter
Body
Data model
Type
MAIL
Forward on Kafka
Language
English-en
English-en subject
Welcome!
Source
Styles
Normal
Hi, #firstName #lastName , this is a notification!
From: #senderName
#senderEmail
body p span
Save
Publish

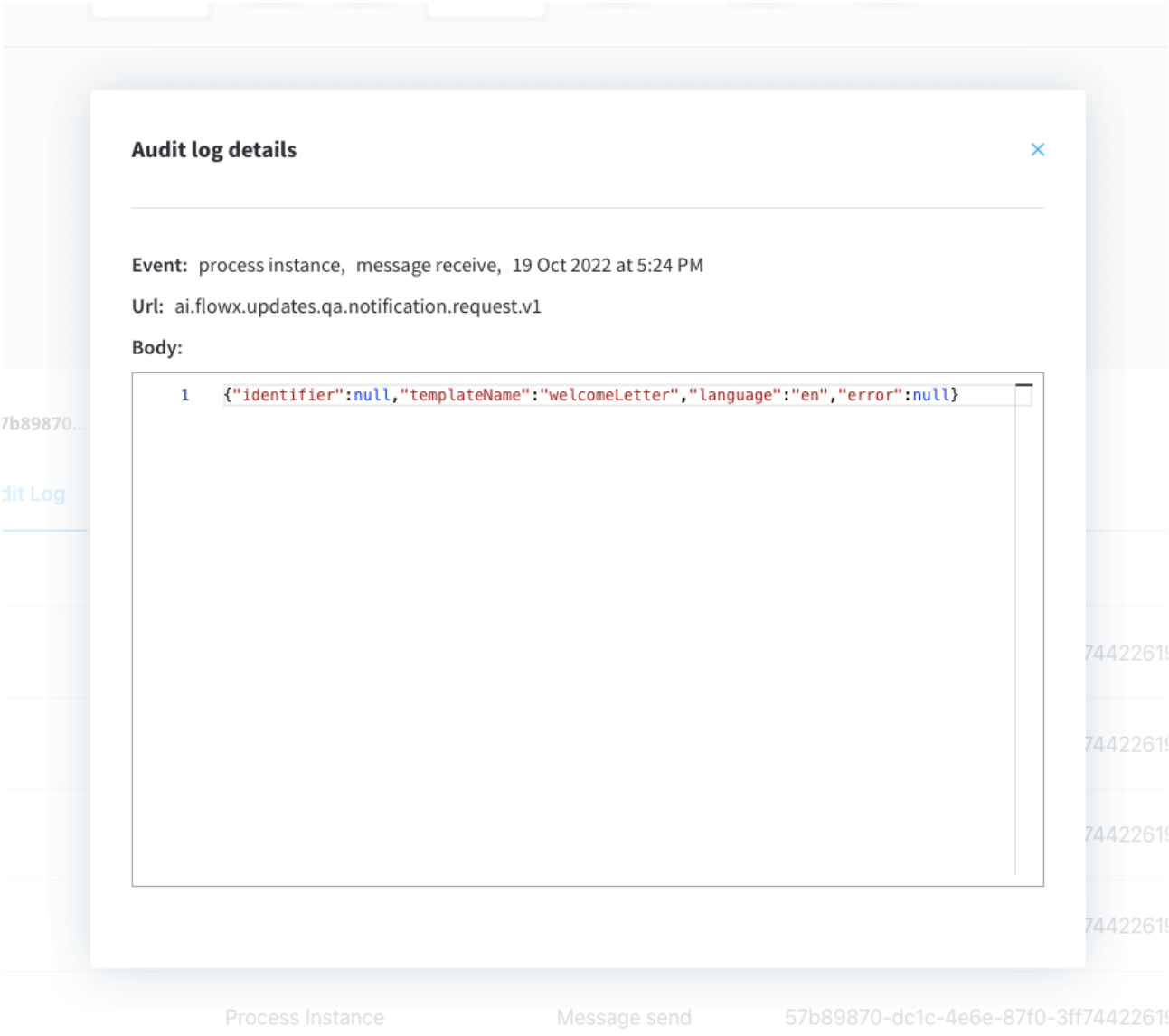


4. Use the FLOWX.AI Designer to create a process definition.
5. Add a **Kafka send event node** and a **Kafka receive event node** (one to send the request, one to receive the reply).
6. Check if the needed topic (defined at the following environment variable) is configured correctly: `KAFKA_TOPIC_NOTIFICATION_INTERNAL_IN`.
7. Add the proper configuration to the action, the Kafka topic, and the body message.



> **INFO**
>
> **Forward on Kafka** option will forward the notification to an external adapter, make sure the needed Kafka topic for forwarding is defined/overwritten using the following environment variable: `KAFKA_TOPIC_EXTERNAL_OUT`.

7. Run the process and look for the response (you can view it via the **Audit log**) or by checking the responses on the Kafka topic



Response example at `KAFKA_TOPIC_NOTIFICATION_INTERNAL_OUT`:

```
{
  "templateName": "welcomeLetter",
  "receivers": [
```

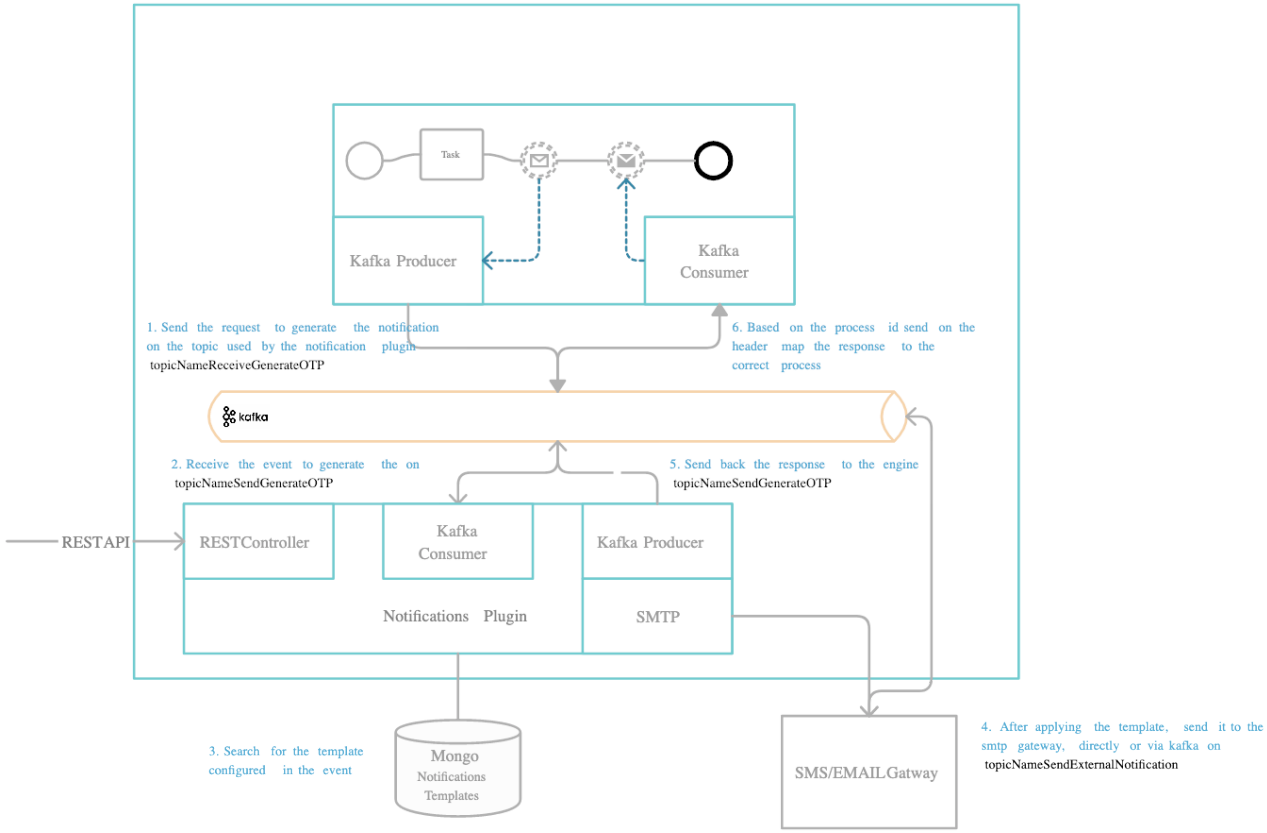
```
    "john.doe@mail.com"
  ],
  "channel": "MAIL",
  "language": "en"
}
```

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Notifications plugin / Using the plugin / OTP flow / Generate OTP

There are some cases when you will need to generate an OTP (One Time Password) from a business flow, for example when validating an email account.

The notifications plugin handles both the actual OTP code generation and sending the code to the user using a defined notification template.



Define needed Kafka topics

Kafka topic names can be set by using environment variables:

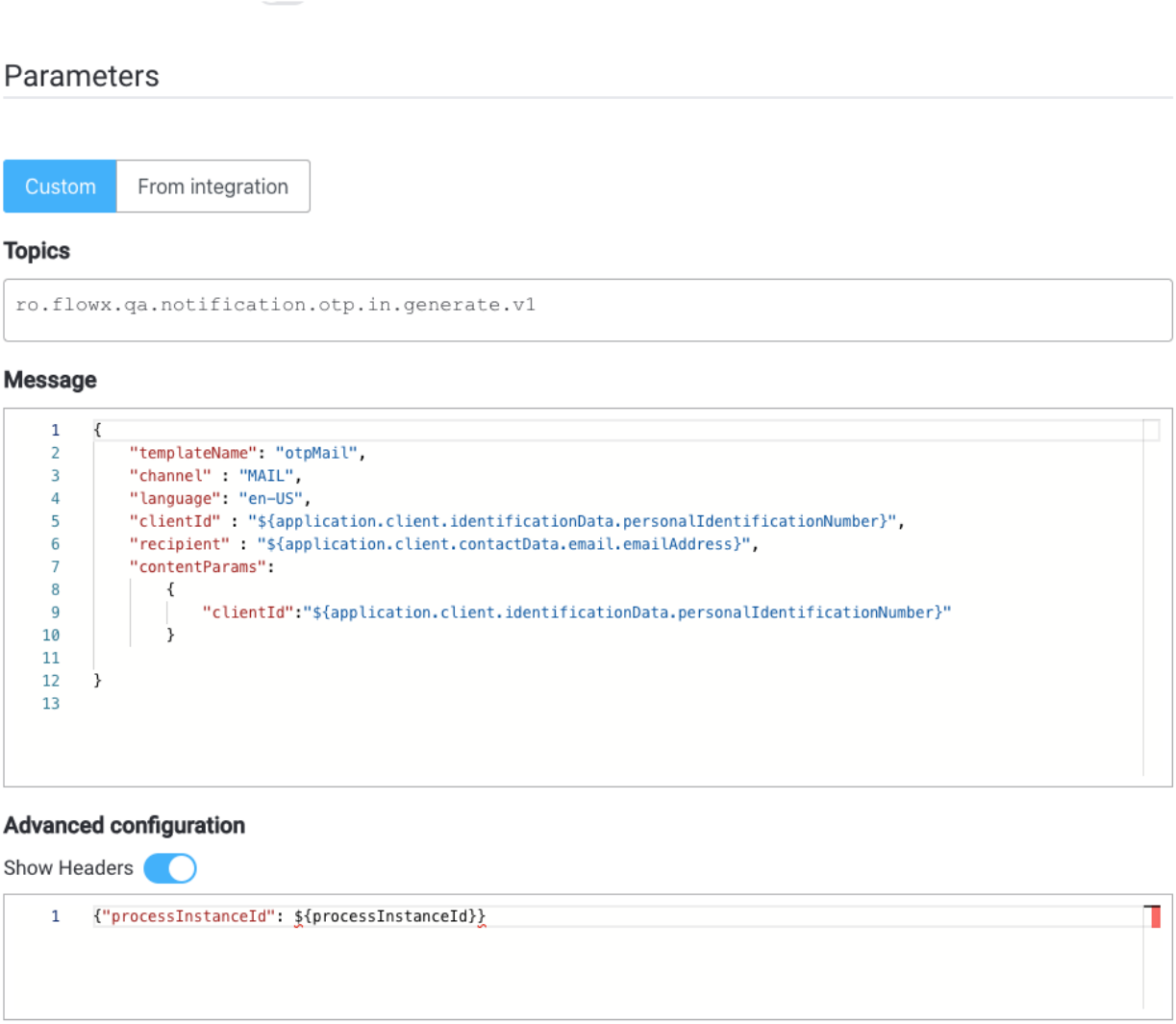
- `KAFKA_TOPIC_OTP_GENERATE_IN`
- `KAFKA_TOPIC_OTP_GENERATE_OUT` - after the OTP is generated and sent to the user, this is the topic used to send the response back to the Engine.

> **⚠ CAUTION**
>
> The Engine is listening for messages on topics with names of a certain pattern, make sure to use an outgoing topic name that matches the pattern configured in the Engine.

Request to generate an OTP

Values expected in the request body:

- templateName: the name of the notification template that is used (created using the WYSIWYG editor)
- channel: notification channel: SMS / MAIL
- recipient: notification receiver: email / phone number
- notification template content parameters (for example, clientId): parameters that should be replaced in the notification template



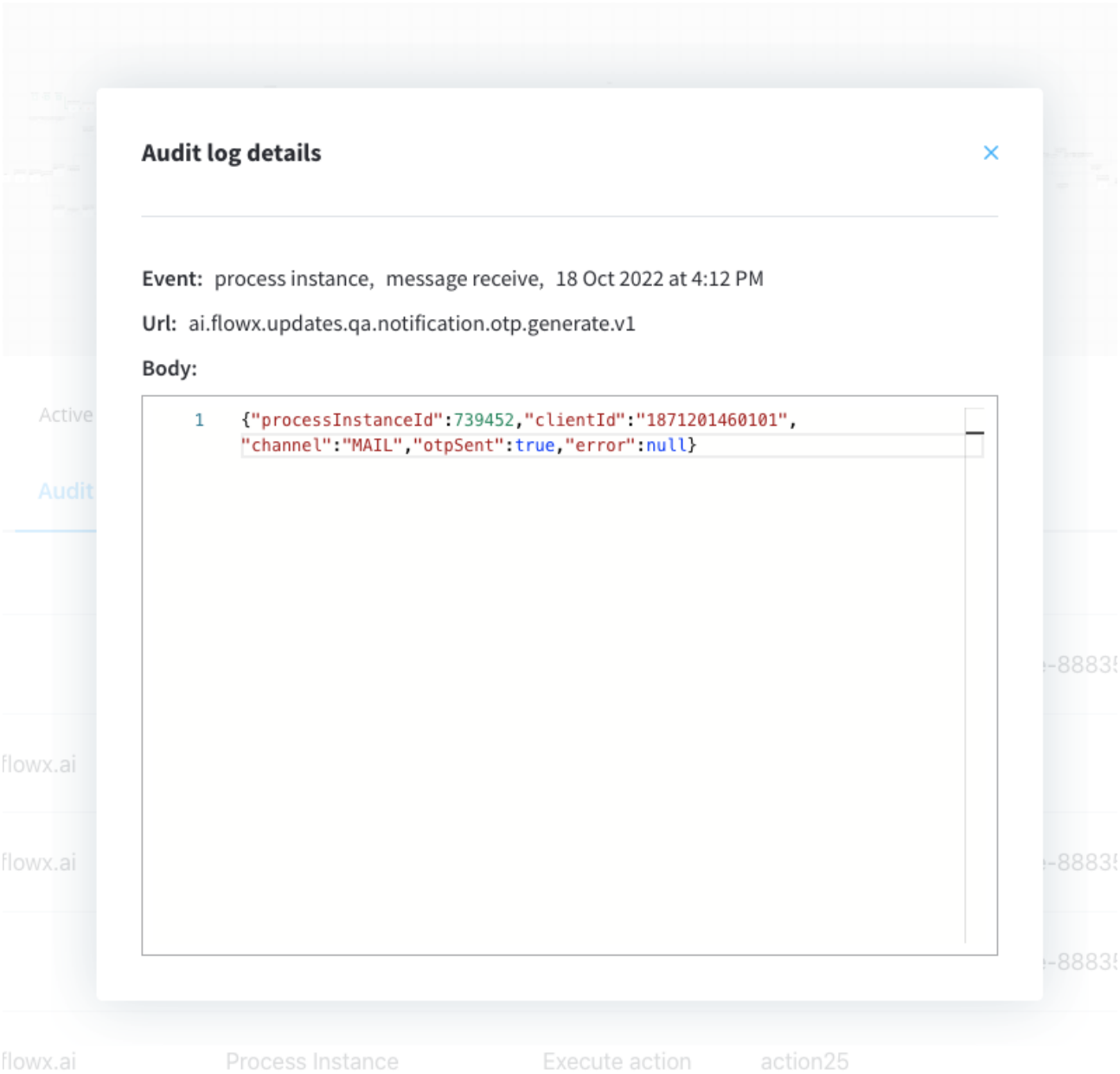
Response from generate OTP

Values expected in the reply body:

- processInstanceId = process instance ID
- clientId = the client id (in this case the SSN number of the client)
- channel = notification channel used
- otpSent = confirmation if the notification was sent: true or false

- error = error description, if any

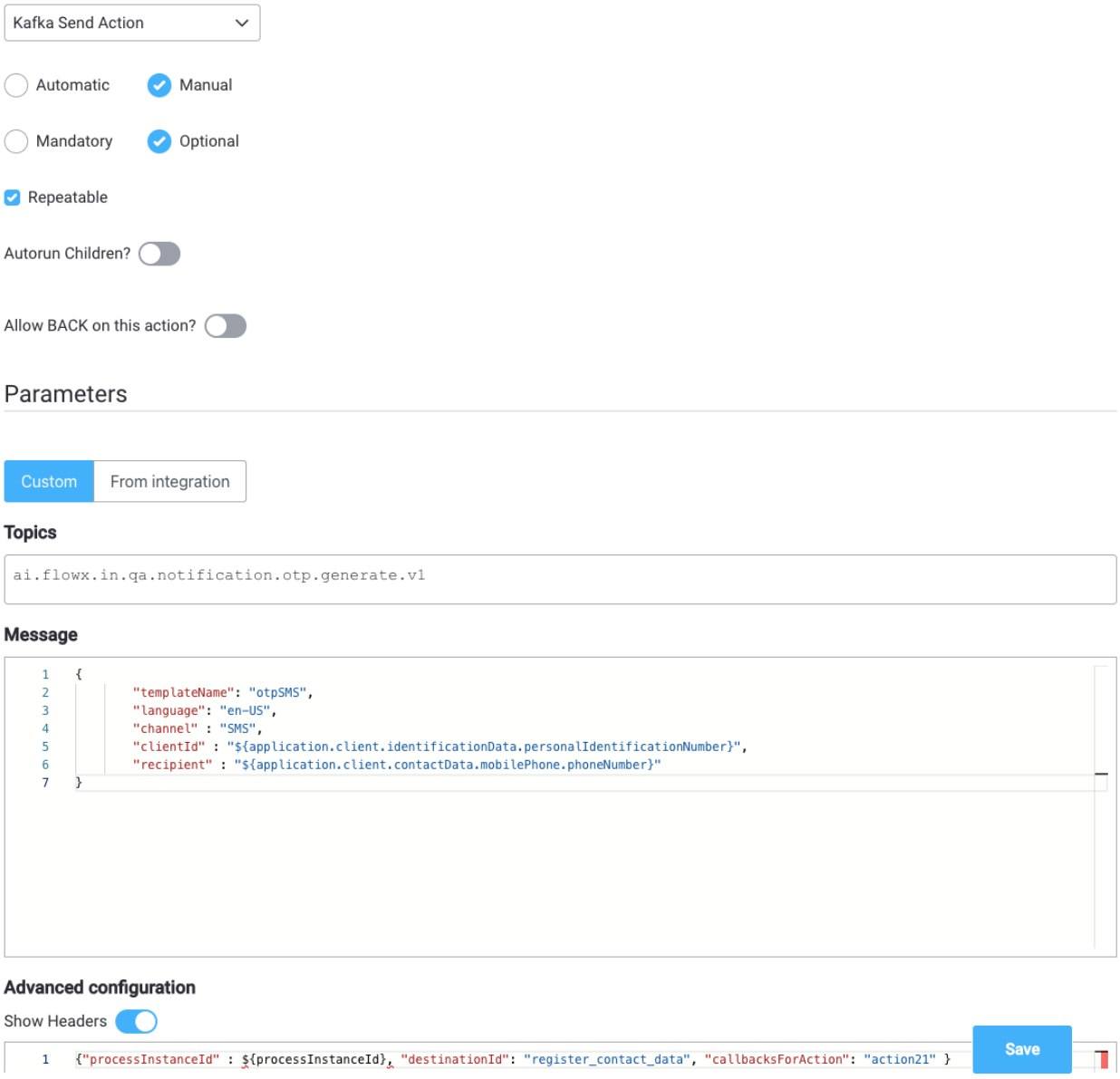
Example:



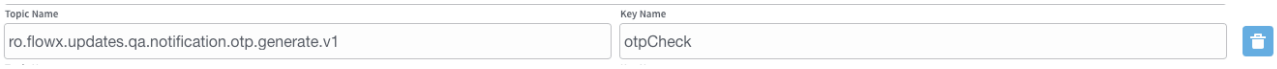
Example: generate an OTP from a business flow

It is important to identify what is the business identifier that you are going to use to validate that OTP, it can be, for example, a user identification number.

1. Configure the templates that you want to use (for example, an SMS template).
2. Check that the needed topics are configured correctly: the topic used to generate OTP (`KAFKA_TOPIC_OTP_GENERATE_IN`) and the topic used to receive the response (`KAFKA_TOPIC_OTP_GENERATE_OUT`).
3. Use the FLOWX.AI Designer to add a new Kafka send event to the correct node in the process definition.
4. Add the proper configuration to the action, the Kafka topic, and configure the body message.



5. Add a node to the process definition (for the Kafka receive event).
6. Configure on what key you want to receive the response on the process instance params.



Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Notifications plugin / Using the plugin / OTP flow / Validate OTP

Define needed Kafka topics

Kafka topic names can be set by using environment variables:

- `KAFKA_TOPIC_OTP_VALIDATE_IN` - the event sent on this topic (with an OTP and an identifier) will check if the OTP is valid
- `KAFKA_TOPIC_OTP_VALIDATE_OUT` - the response for this request will validate an OTP, the reply is sent back to the Engine on this topic

> ⚠ **CAUTION**
>
> The Engine is listening for messages on topics with names of a certain pattern, make sure to use an outgoing topic name that matches the pattern configured in the Engine.

Request to validate an OTP

Values expected in the request body:

- processInstanceId = process instance ID

- client id = the user unique ID in the system
- channel = notification channel: SMS/MAIL
- otp = OTP code that you received, used to compare with the one that was sent from the system

Example:

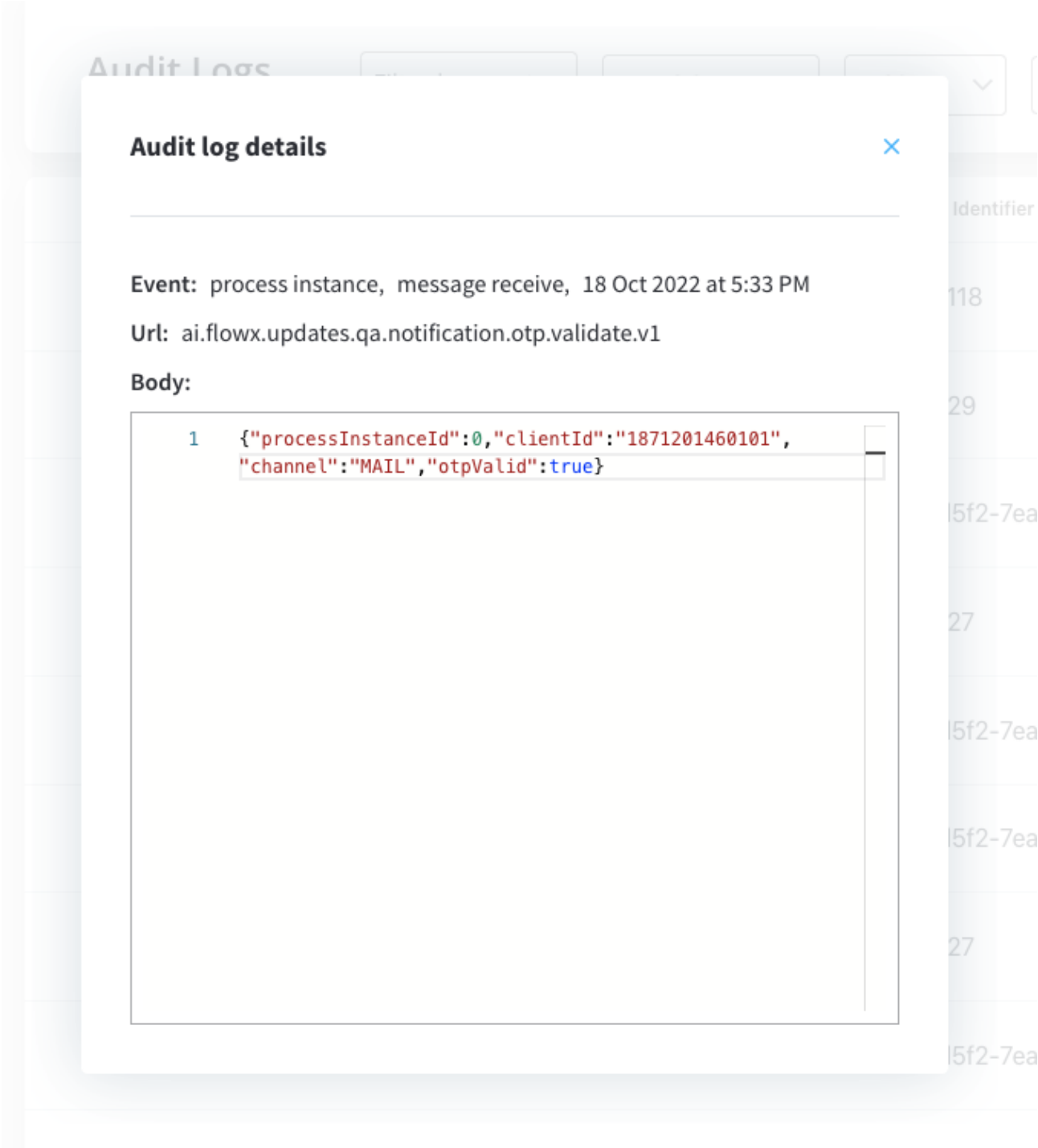
```
{
    "processInstanceId": 12345,
    "clientId": "1871201460101",
    "channel": "MAIL",
    "otp": "1111"
}
```

Reply from validate OTP

Values expected in the reply body:

- client id = the user unique id in the system
- channel = notification channel used
- otpValid = confirmation if the provided OTP code was the same as the one sent from the system

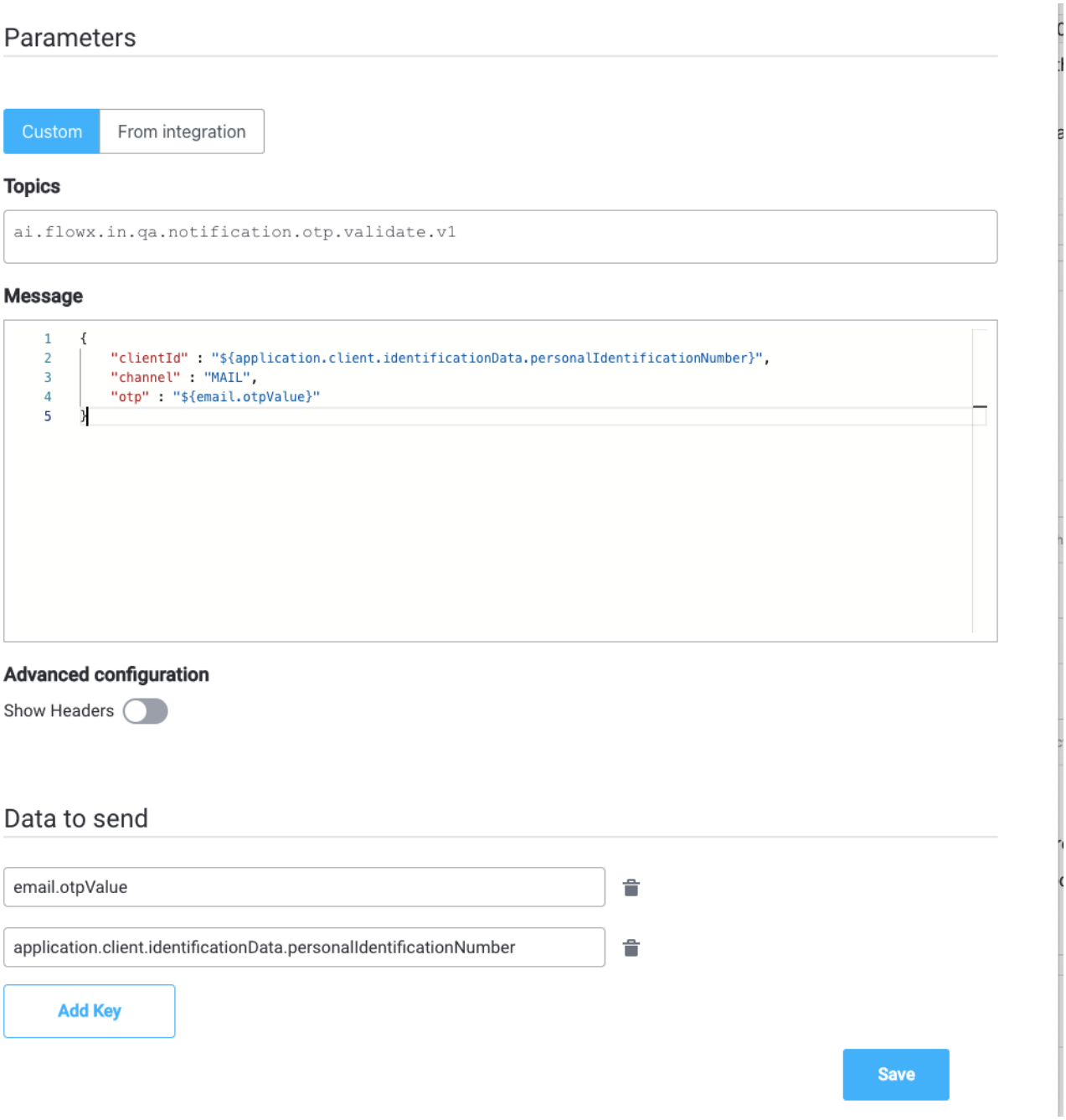
Example:



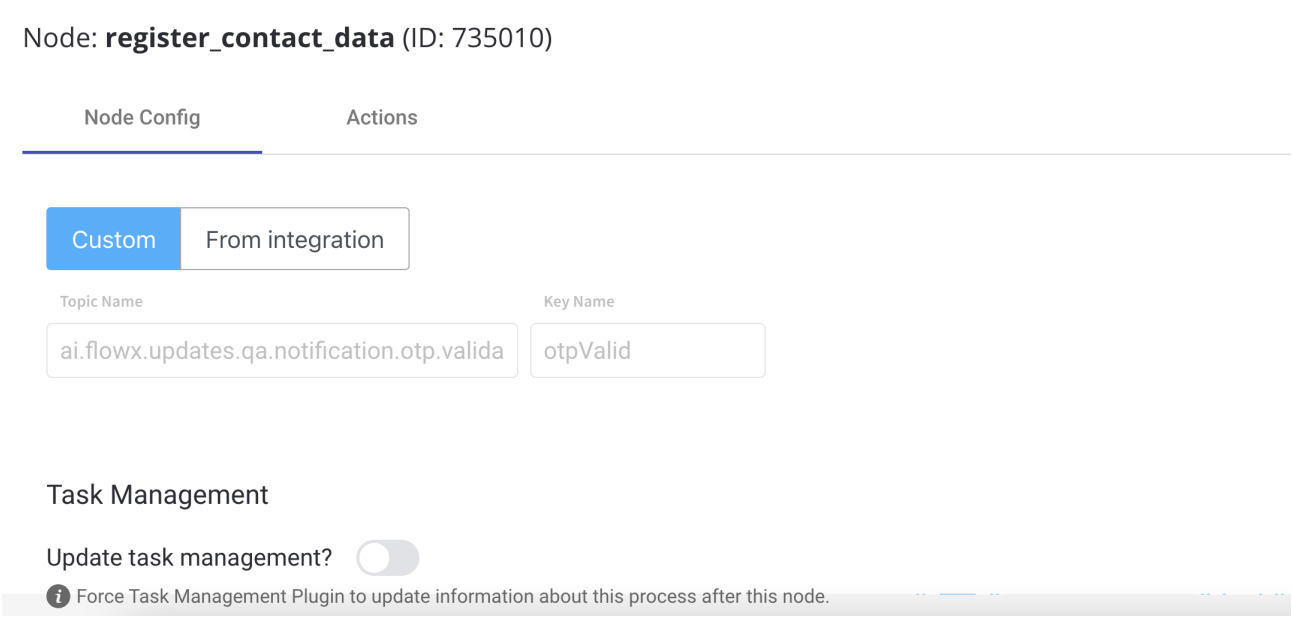
Example: validate an OTP from a business flow

Similar to the generation of the OTP you can validate the OTP that was generated for an identifier.

1. Check that the needed topics are configured correctly: (`KAFKA_TOPIC_OTP_VALIDATE_IN` and `KAFKA_TOPIC_OTP_VALIDATE_OUT`)
2. Add the actions for sending the request to validate the OTP on the node that contains the 'Generate OTP' actions
3. Add the proper configuration to the action, the Kafka topic and configure the body message.



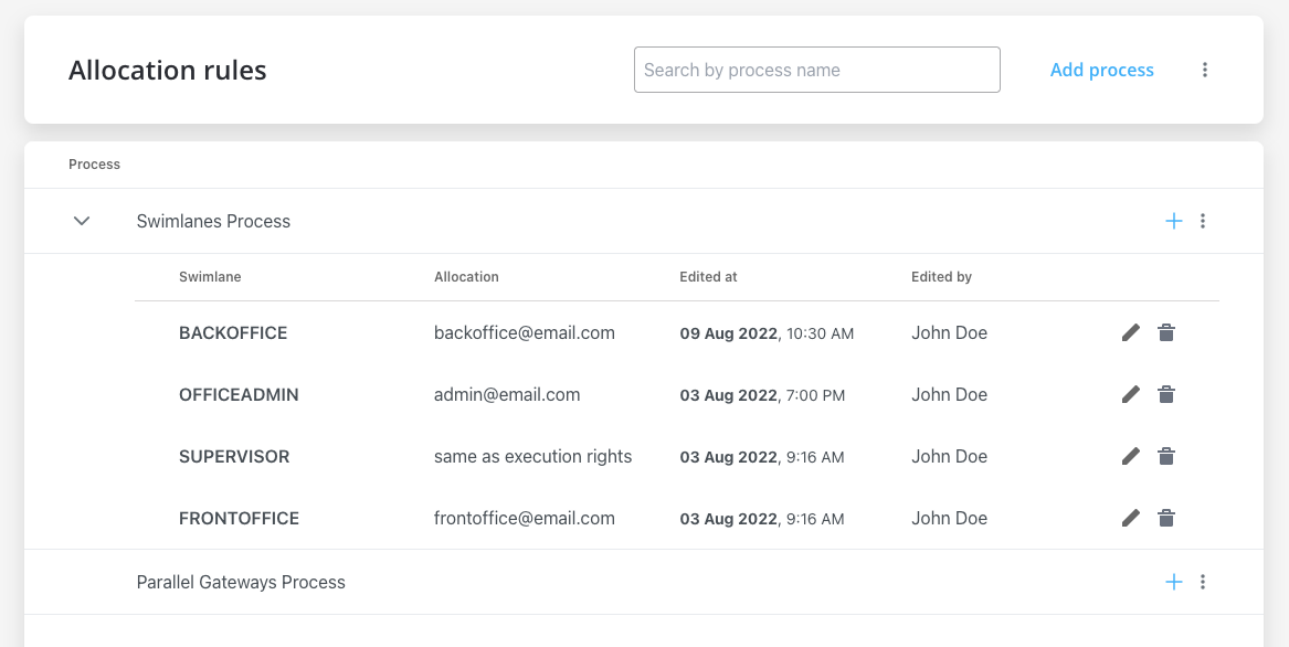
4. Add a node to the process definition (for the Kafka receive event)
5. Configure on what key you want to receive the response on the process instance parameters



Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Task management / Using allocation rules

Allocation rules are meant to define when tasks should be auto-assigned to users when they reach a swimlane that has a specific role configured (for example, specific tasks will be assigned for the *front office* and specific tasks for the *back office* only).

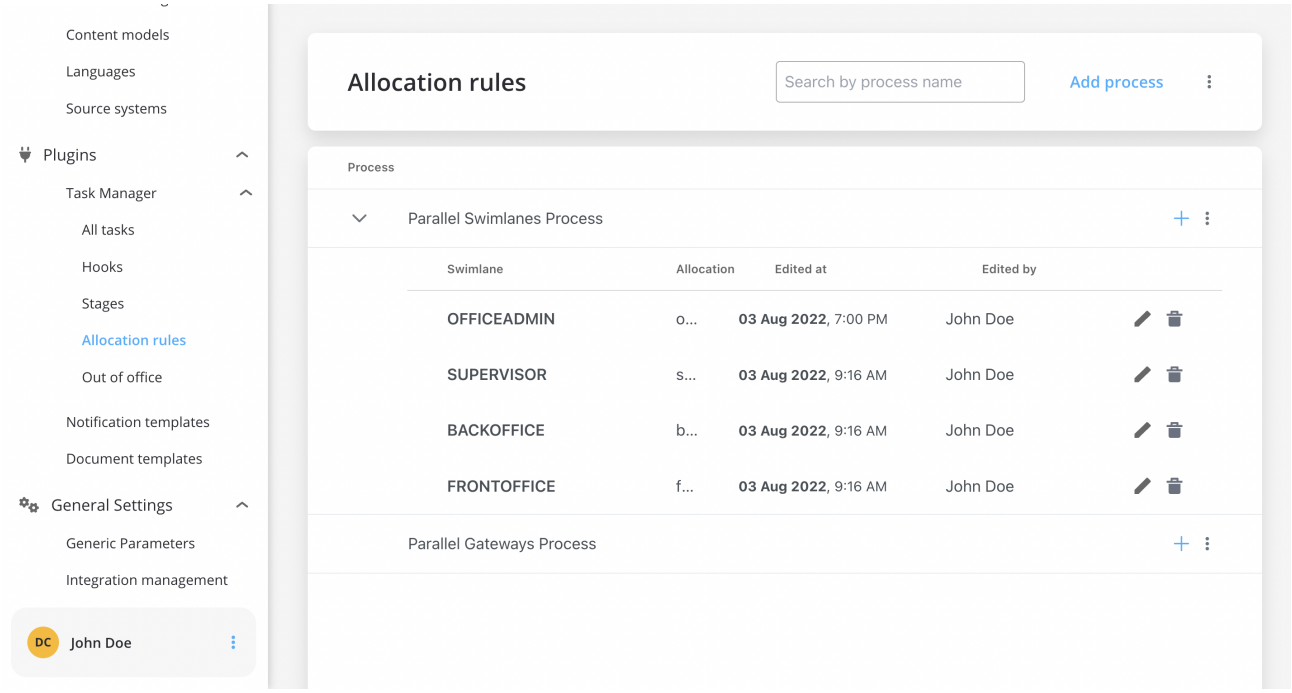


> **INFO**
>
> Tasks will always be allocated depending on the users load (number of tasks) from current/other processes. If there are two or more users with the same number of assigned tasks, the task will be randomly assigned to one of them.

Accessing allocation rules

To access the allocation rules, follow the next steps:

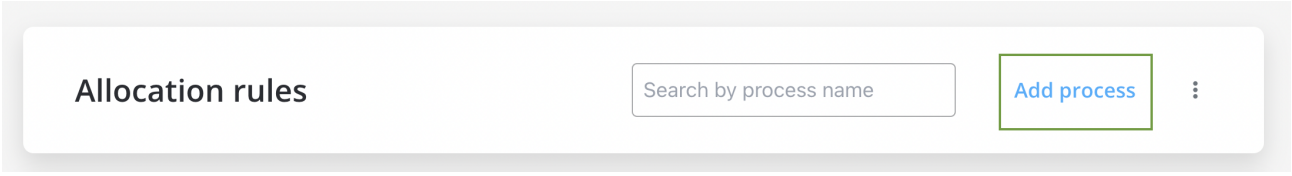
1. Open
   The fallback content to display on prerendering
   .
2. From the side menu, under **Task Management**, select the **Allocation rules** entry.



Adding process and allocation rules

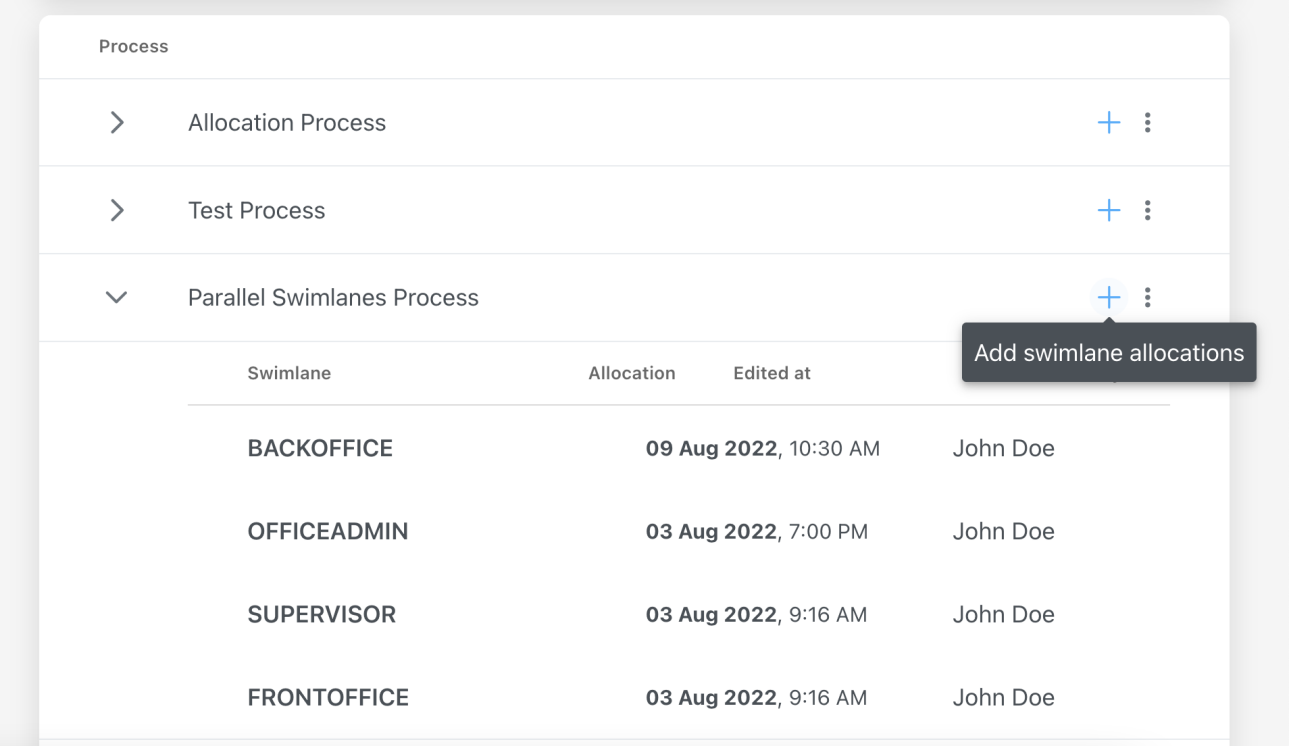
To add process and allocation rules, follow the next steps:

1. Click **Add process** button, in the top-right corner. More details on how to create/configure a process are here.



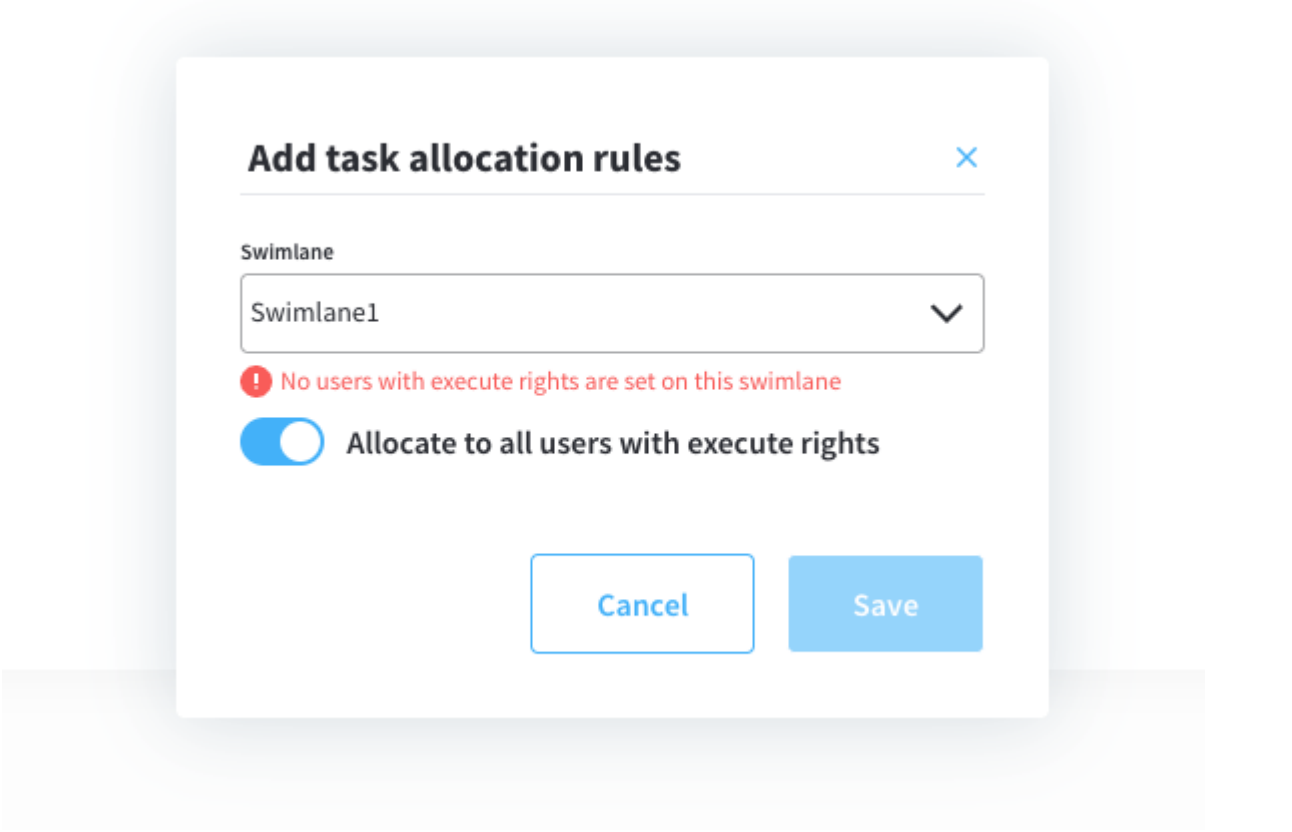
2. Select a **process definition** from the drop-down list.

3. Click **Add swimlane allocations button (+)** to add allocations.

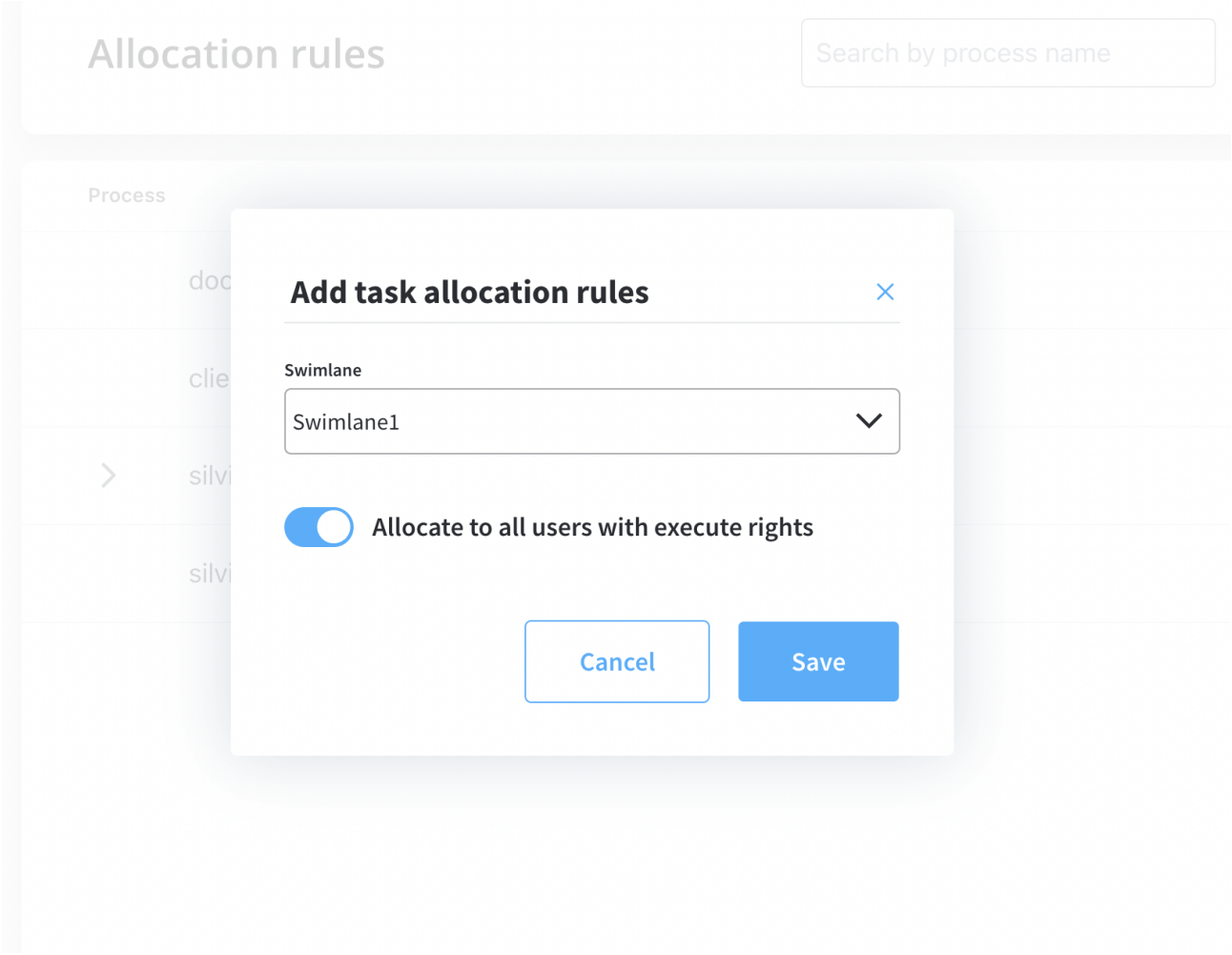


CAUTION

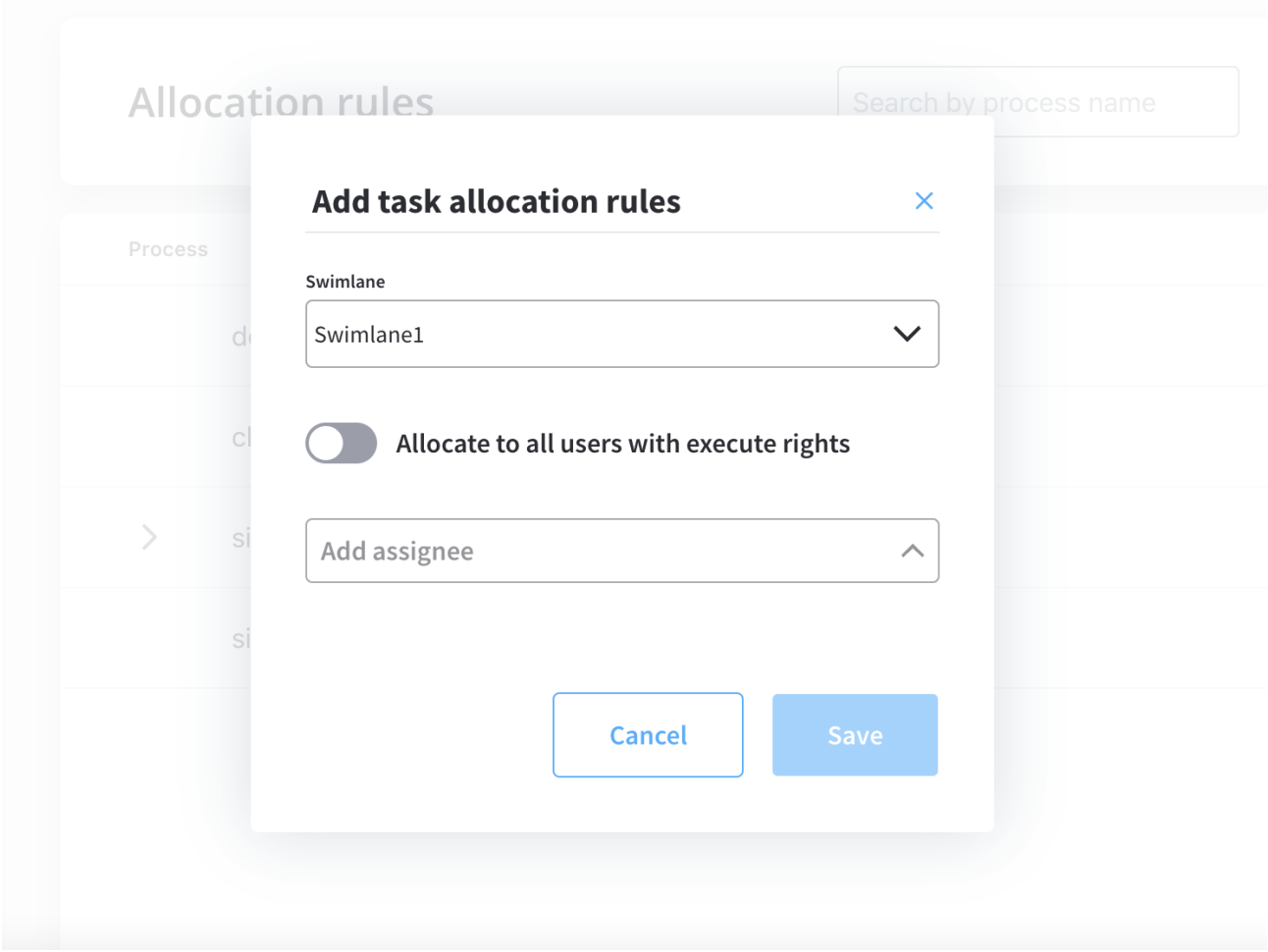
NOTE! If there are no users with execute rights in the swimlane you want to add (`hasExecute: false`), the following error message will be displayed:



4. **Option 1**: Allocate all users with `execute rights`.



5. **Option 2**: Allocate only users you choose from the drop-down list. You can use the search function to filter users by name.



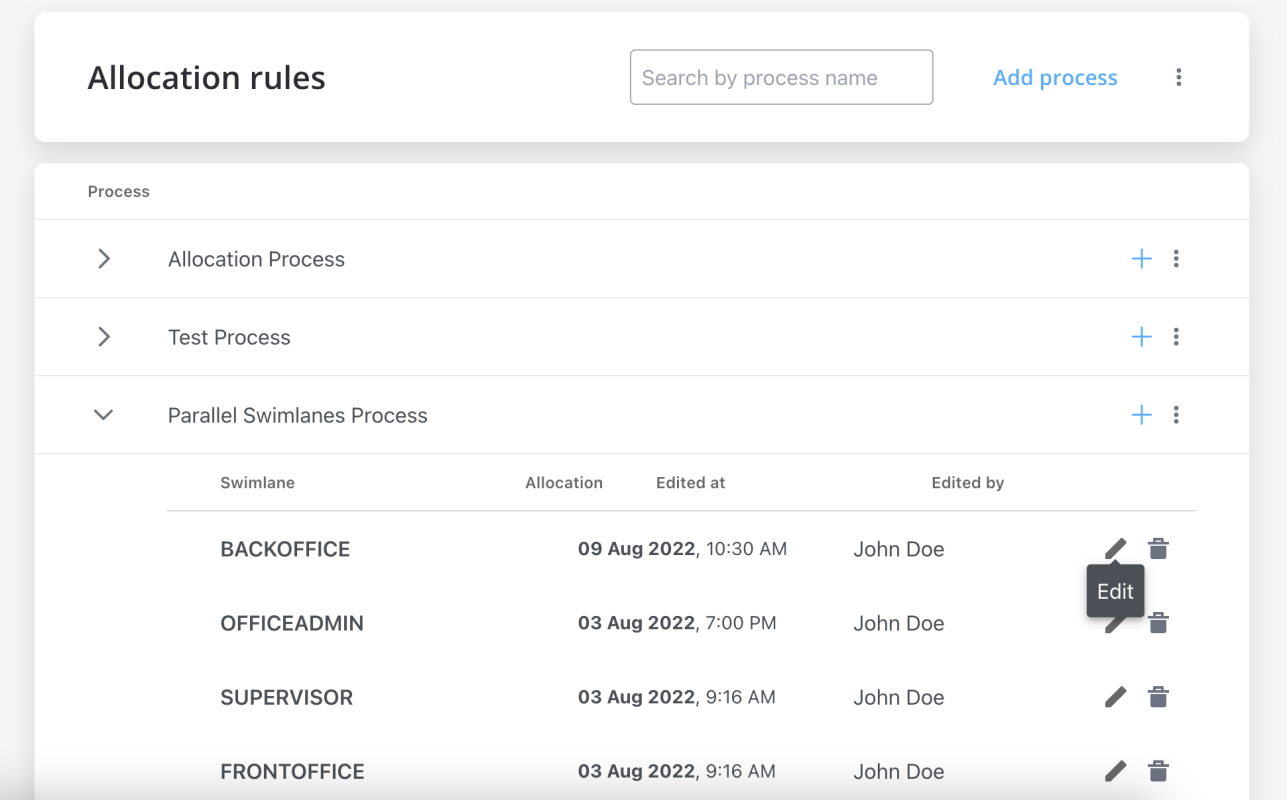
6. Click **Save**.

> **INFO**
>
> Users with out-of-office status will be skipped by automatic allocation. More information about out-of-office feature, here.

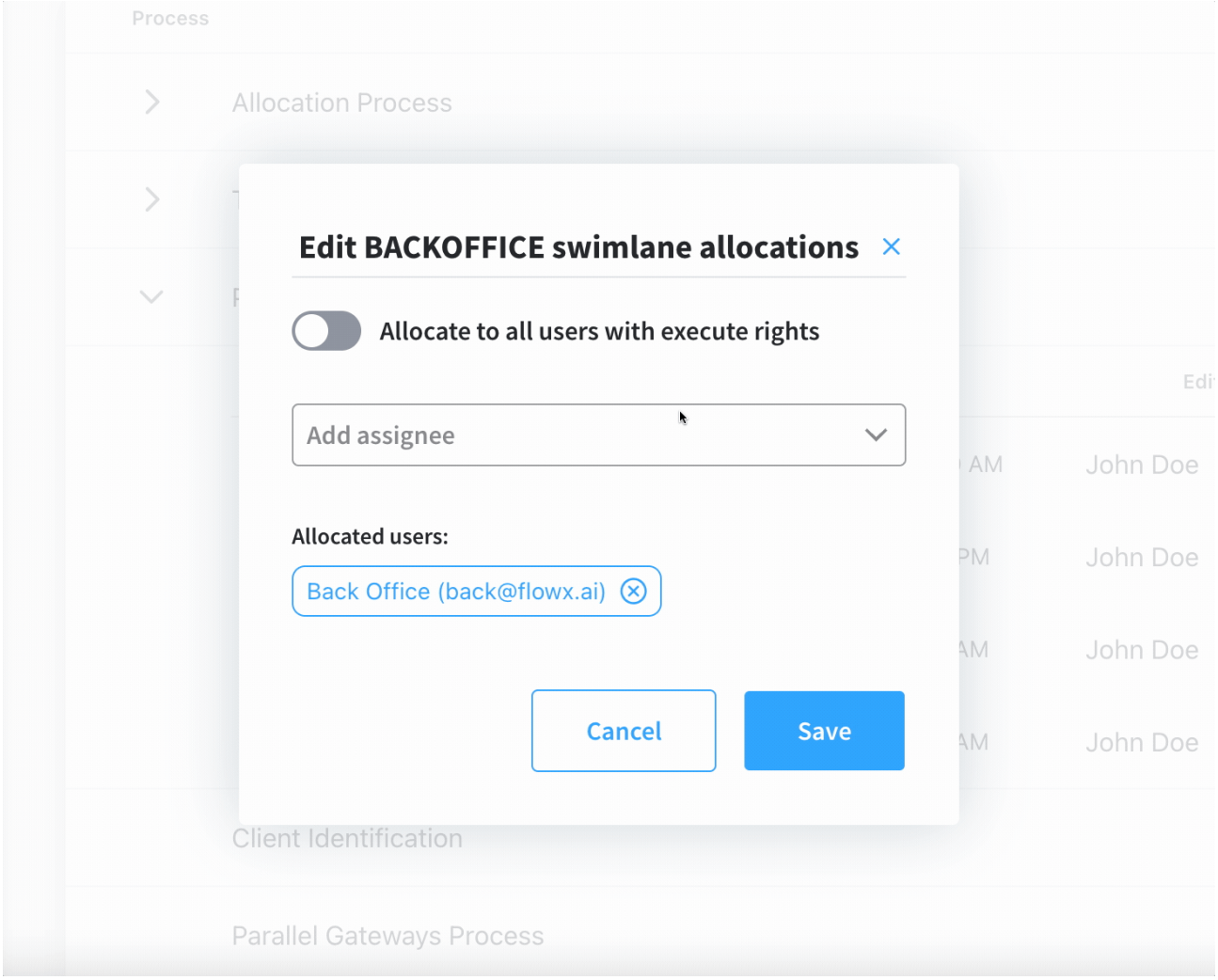
Editing allocation rules

To edit allocation rules, follow the next steps:

1. Click **Edit** button.



2. Change the allocation method.



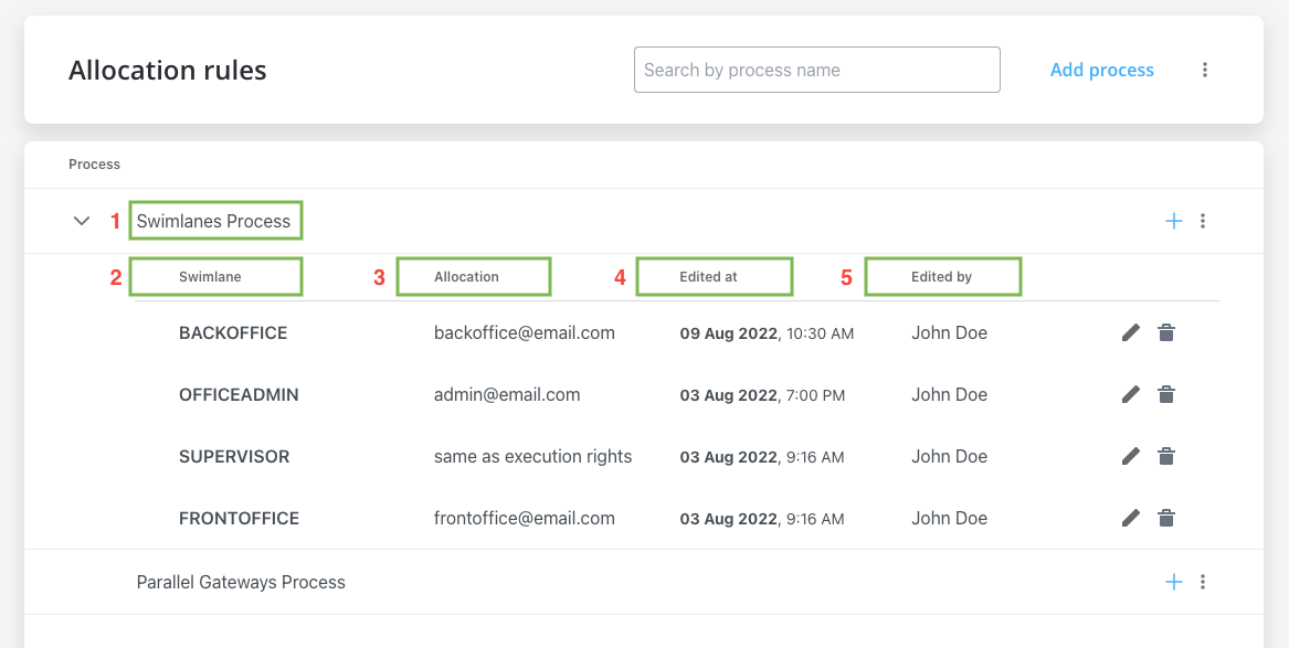
3. Click **Save.**

Viewing allocation rules

The allocation rules list displays all the configured swimlanes grouped by process:

1. **Process** - the process definition name where the swimlanes were configured
2. **Swimlane** - the name of the swimlane
3. **Allocation** - applied allocation rules
4. **Edited at** - the last time when an allocation was edited

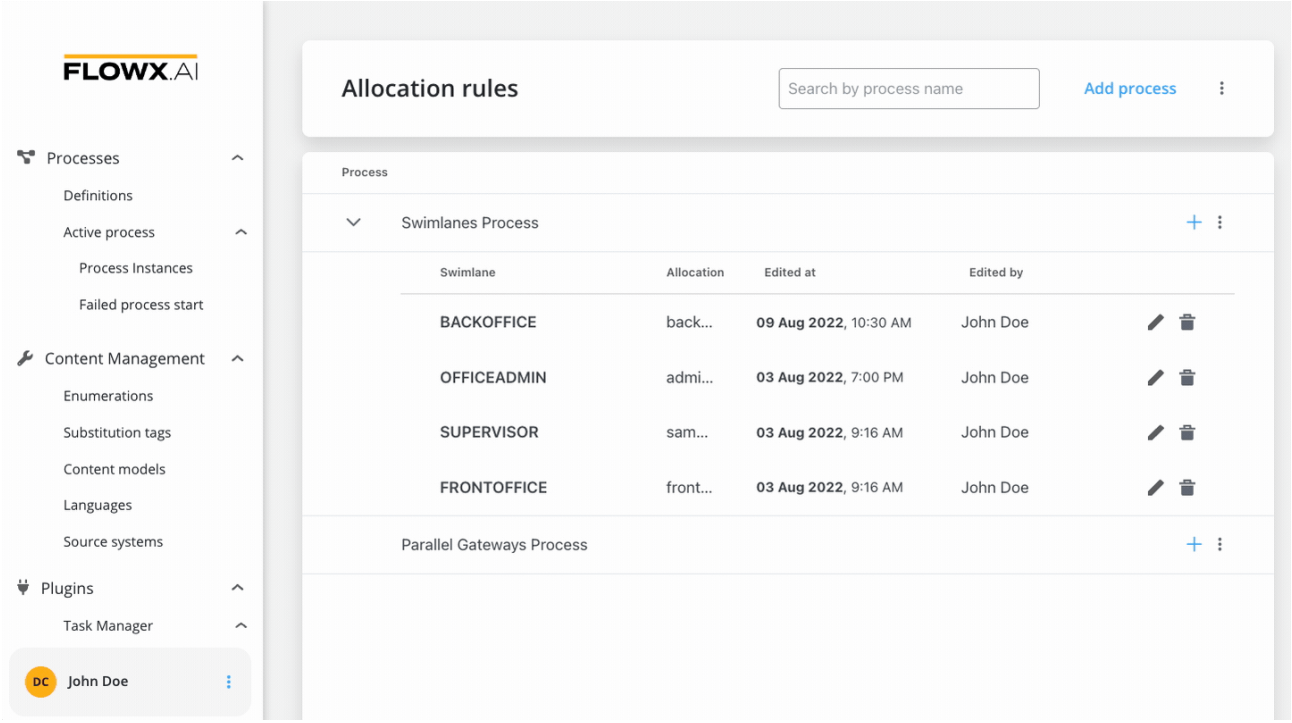
5. **Edited by** - the user who edited/created the allocation rules



Exporting/importing process allocation rules

To copy process allocation rules and move them between different environments, you can use the export/import feature.

You can export process allocation rules as JSON files directly from the allocation rules list:



Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Task management / Using hooks

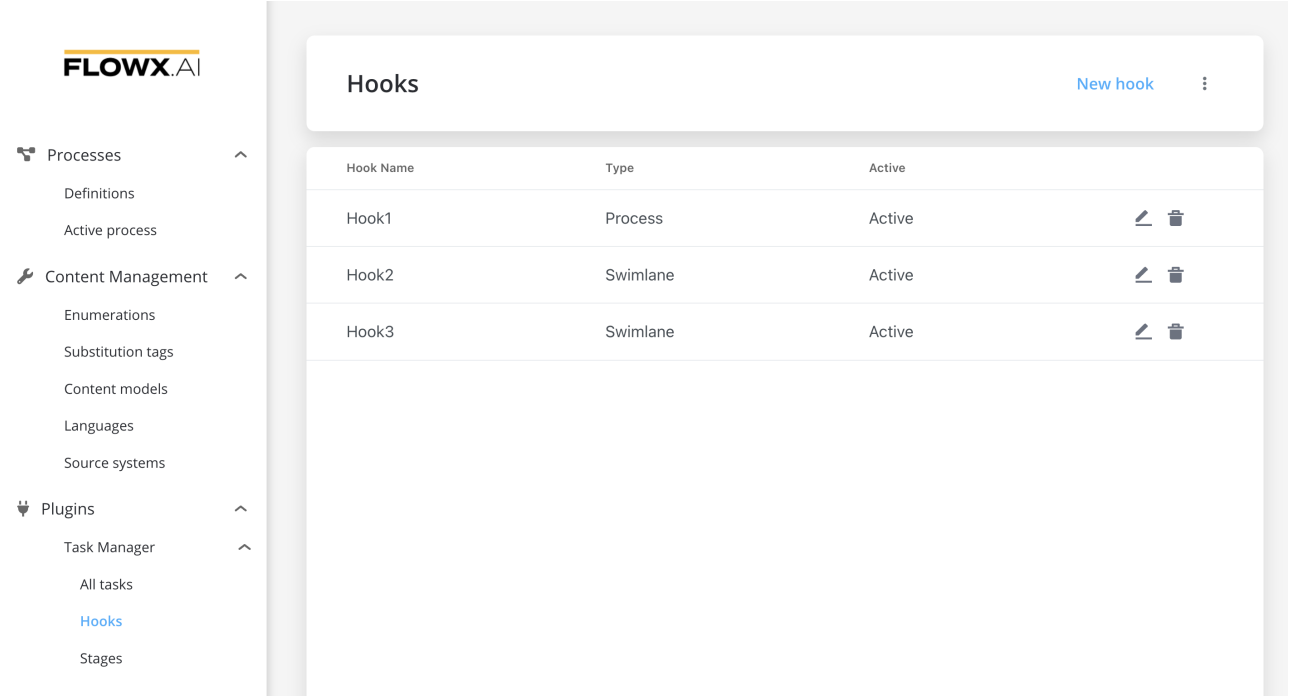
Hooks allow you to extract stateful logic from a component, so it can be tested and reused independently.

Users with task management permissions can create hooks to trigger specific

The fallback content to display on prerendering
, such as sending notifications when

The fallback content to display on prerendering
occur. Follow the instructions below to set up roles for hooks scope usage:

» Manage hooks roles



Hooks can be linked to different events and define what will happen when they are triggered. Below you can find a list of all possible triggers for each hook.

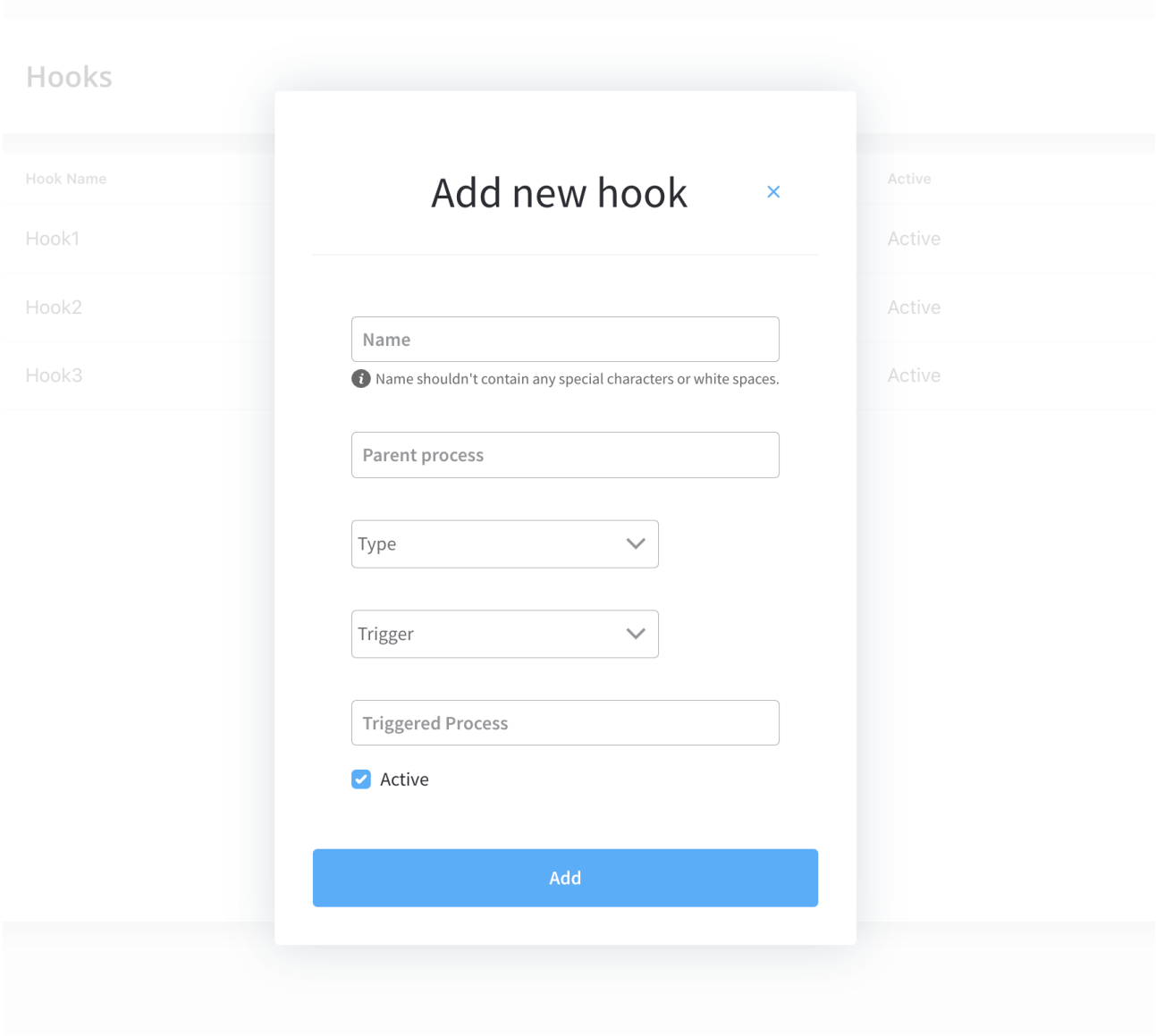
Process **Swimlane** **Stage**

- unique result
- only one rule will match, or no rule

Creating a hook

To create a new hook, follow the next steps:

1. Open
   The fallback content to display on prerendering
   .
2. Go to Task Manager and select **Hooks**.
3. Click **New Hook** (you can also import or export a hook).
4. Fill in the required details.

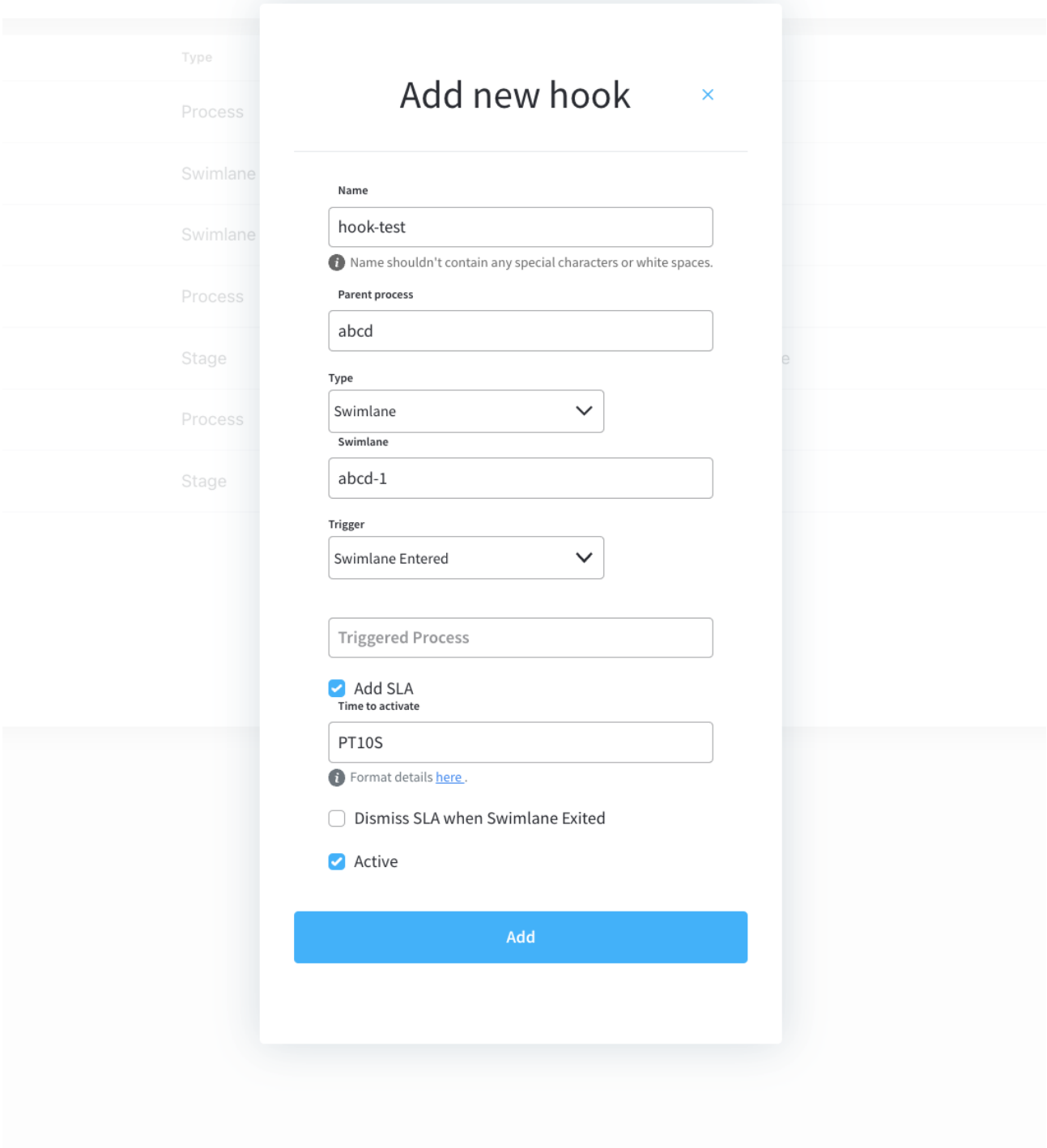


Types of hooks

There are three types of hooks you can create in Task Manager:

- process hooks
- swimlane hooks
- stage hooks

> **INFO**
>
> Swimlane and stage hooks can be configured with an SLA (time when a triggered process is activated).


Add new hook
Name
hook-test
Name shouldn't contain any special characters or white spaces.
Parent process
abcd
Type
Swimlane
Swimlane
abcd-1
Trigger
Swimlane Entered
Triggered Process
Add SLA
Time to activate
PT10S
Format details here .
Dismiss SLA when Swimlane Exited
Active
Add

INFO

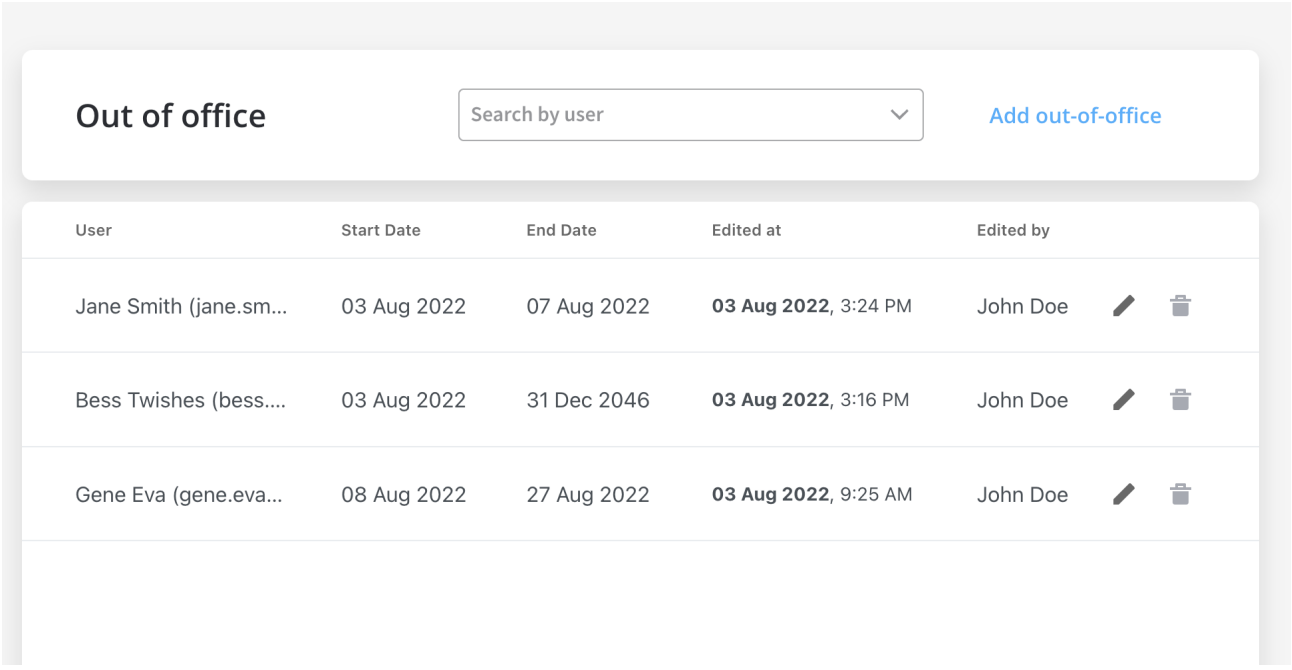
> Dismiss SLA is available only for hooks configured with SLA.

Here you can find more information about the SLA - duration formatting.

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Task management / Using out of office records

The Out-of-office feature allows you to register users availability to perform a task. It can be allocated manually or automatically.

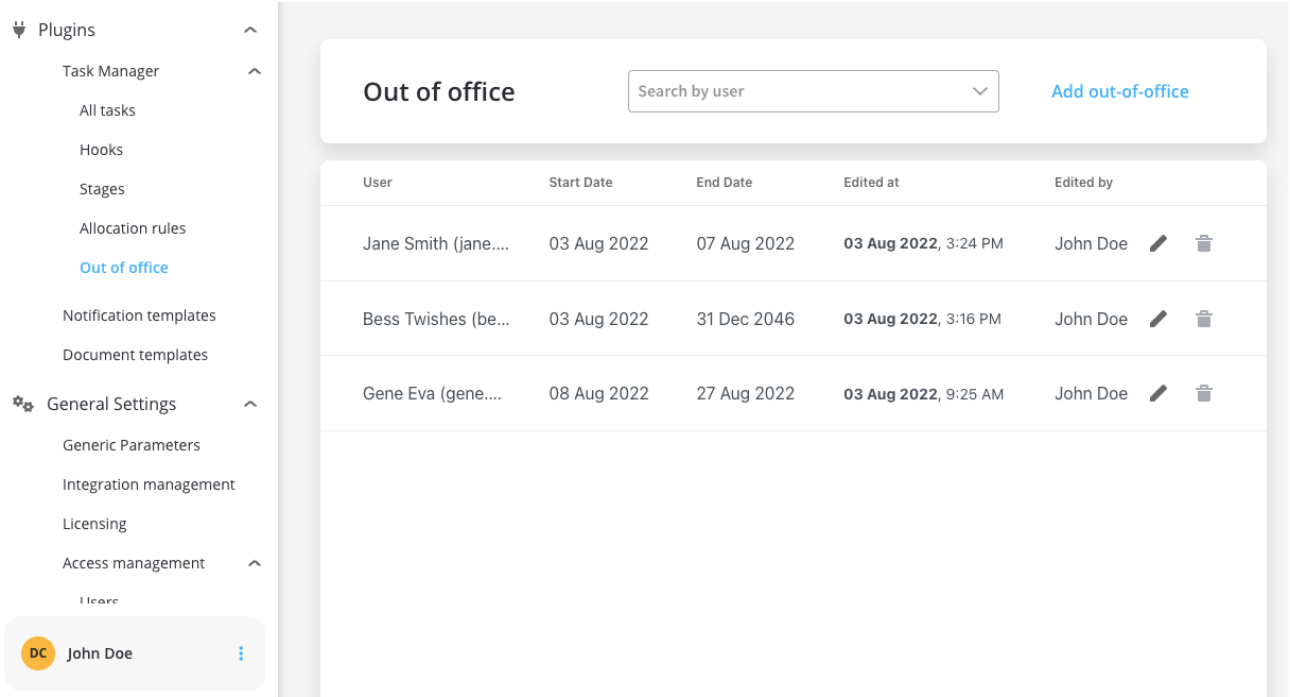


> **INFO**
>
> Users with out-of-office status are excluded from the candidates for automatic task allocation list during the out-of-office period. More information about allocation rules, here.

Accessing out-of-office records

To add out-of-office records, follow the next steps:

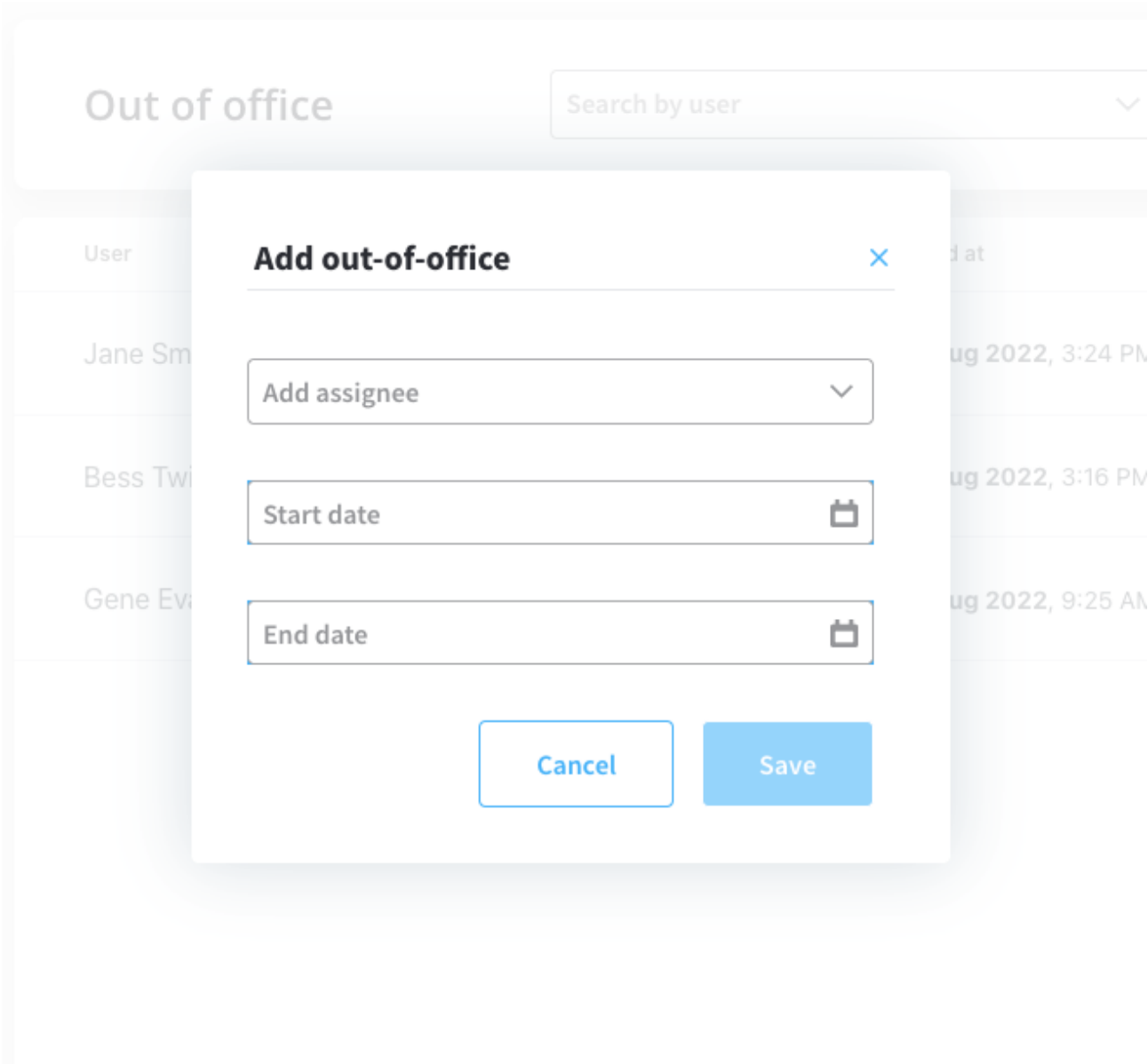
1. Open
   The fallback content to display on prerendering
   .
2. From the side menu, under **Task Management,** select the **Out office entry**.



Adding out-of-office records

To add out-of-office records, follow the next steps:

1. Click **Add out-of-office** button, in the top-right corner.
2. Fill in the following mandatory details:
    - Assignee - user single select
    - Start Date (❗ cannot be earlier than tomorrow)
    - End Date (❗ cannot be earlier than tomorrow)

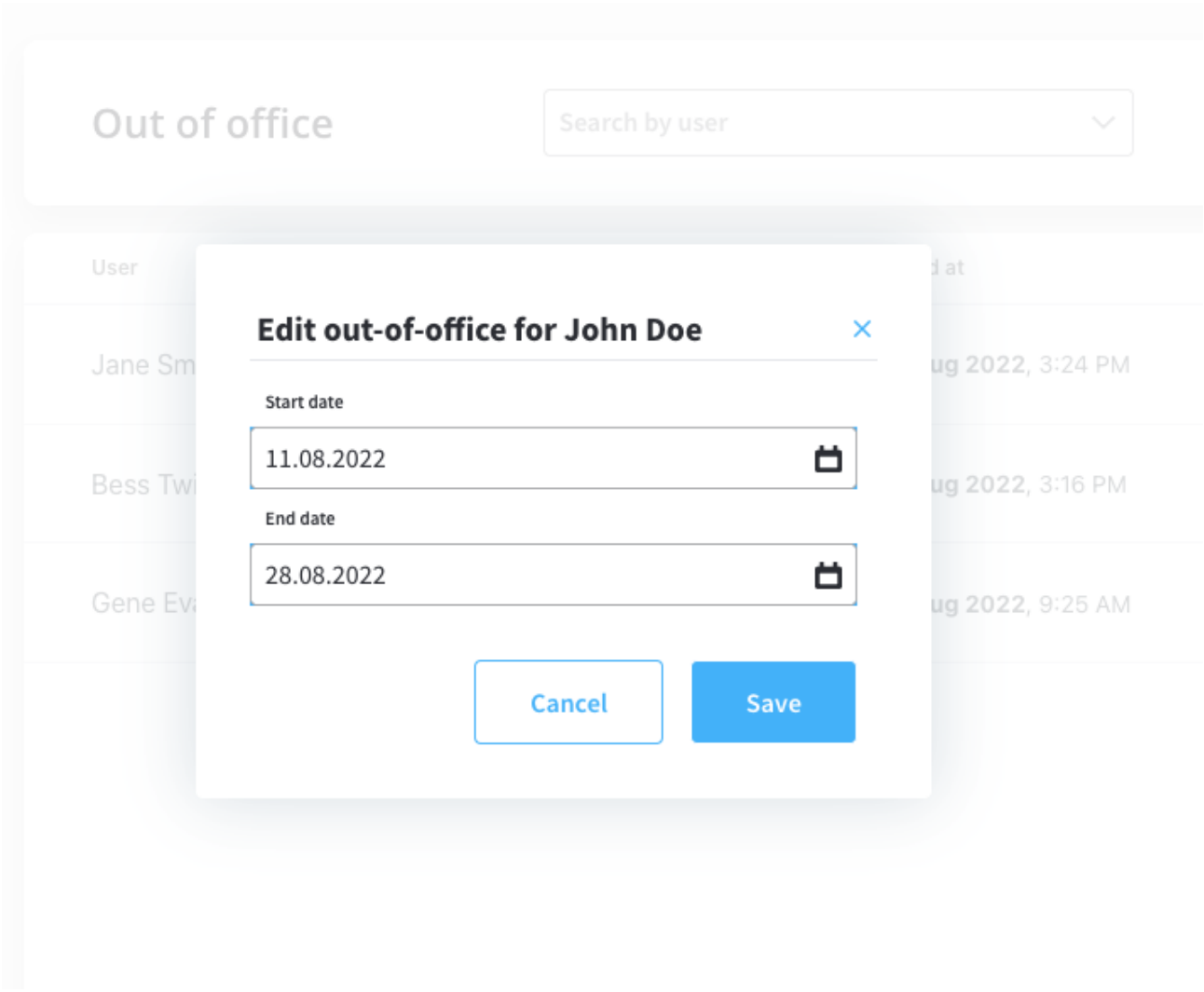


3. Click **Save**.

Editing out-of-office records

To edit out-of-office records, follow the next steps:

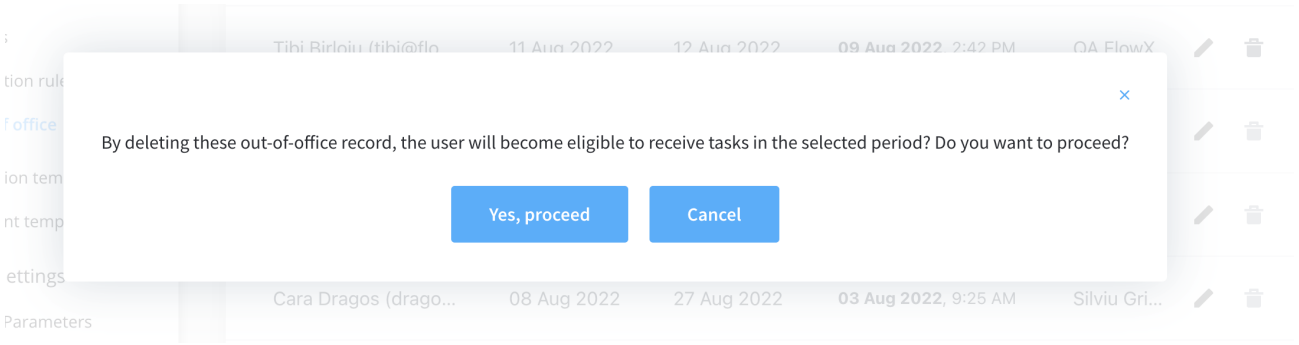
1. Click **Edit** button.
2. Modify the dates (❗ cannot be earlier than tomorrow).
3. Click **Save**.



Deleting out-of-office records

To delete out-of-office records, follow the next steps:

1. From the **out-of-office list**, select a **record**.
2. Click **Delete** button. A pop-up message will be displayed: *"By deleting this out-of-office record, the user will become eligible to receive tasks in the selected period. Do you want to proceed?"*



DANGER

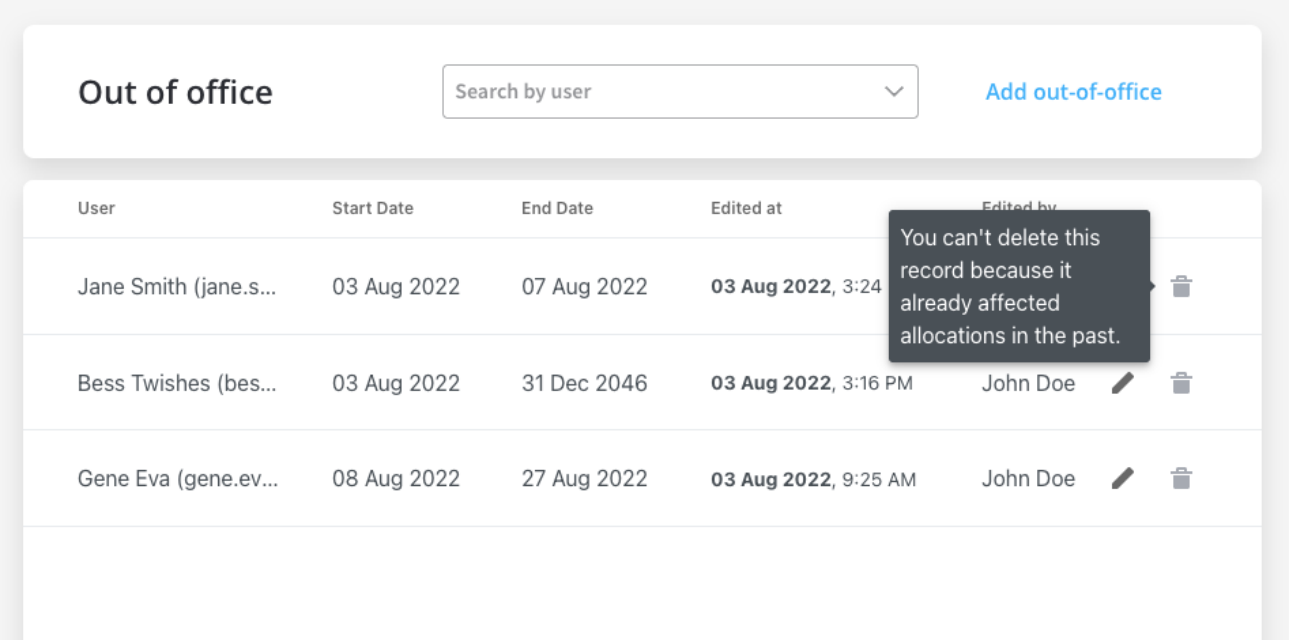
If you choose to delete an out-of-office record, the user is eligible to receive tasks allocation during the mentioned period. More information about automatic task allocation, here.

3. Click **Yes, proceed** if you want to delete the record, click **Cancel** if you want to abort the deletion.

CAUTION

If the out-of-office period contains days selected in the past, the user cannot delete the record, the following message is displayed: *“You can’t delete this*

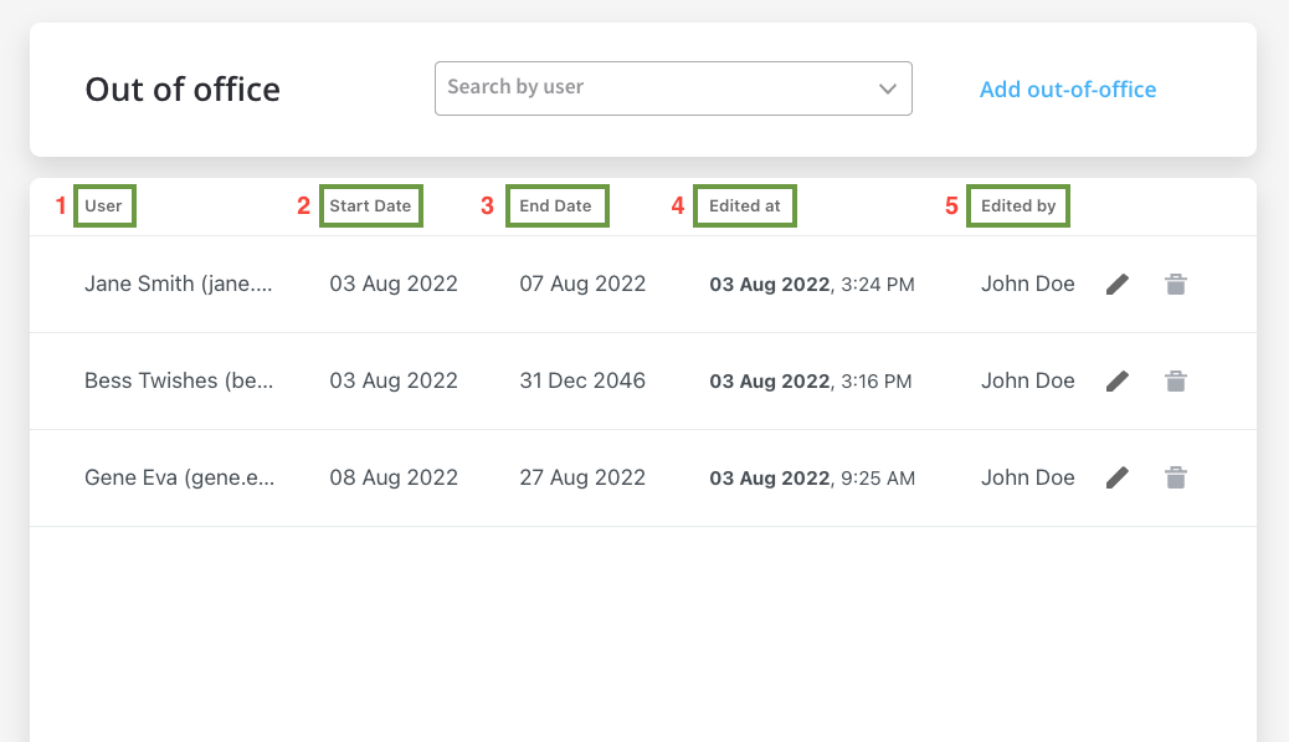
> *record because it already affected allocations in the past. Try to shorten the period, if it didn't end."*



Viewing out-of-office records

The out-of-office records list contains the following elements:

1. **User** - firstName, lastName, userName
2. **Start Date** - the date when the out-of-office period will be effective
3. **End Date** - the date when the out-of-office period will end
4. **Edited at** - the last time when an out-of-office record was edited
5. **Edited by** - the user who edited/created the out-of-office record



INFO

The list is sorted in reverse chronological order by “edited at” `dateTime` (newest added on top).

Was this page helpful?

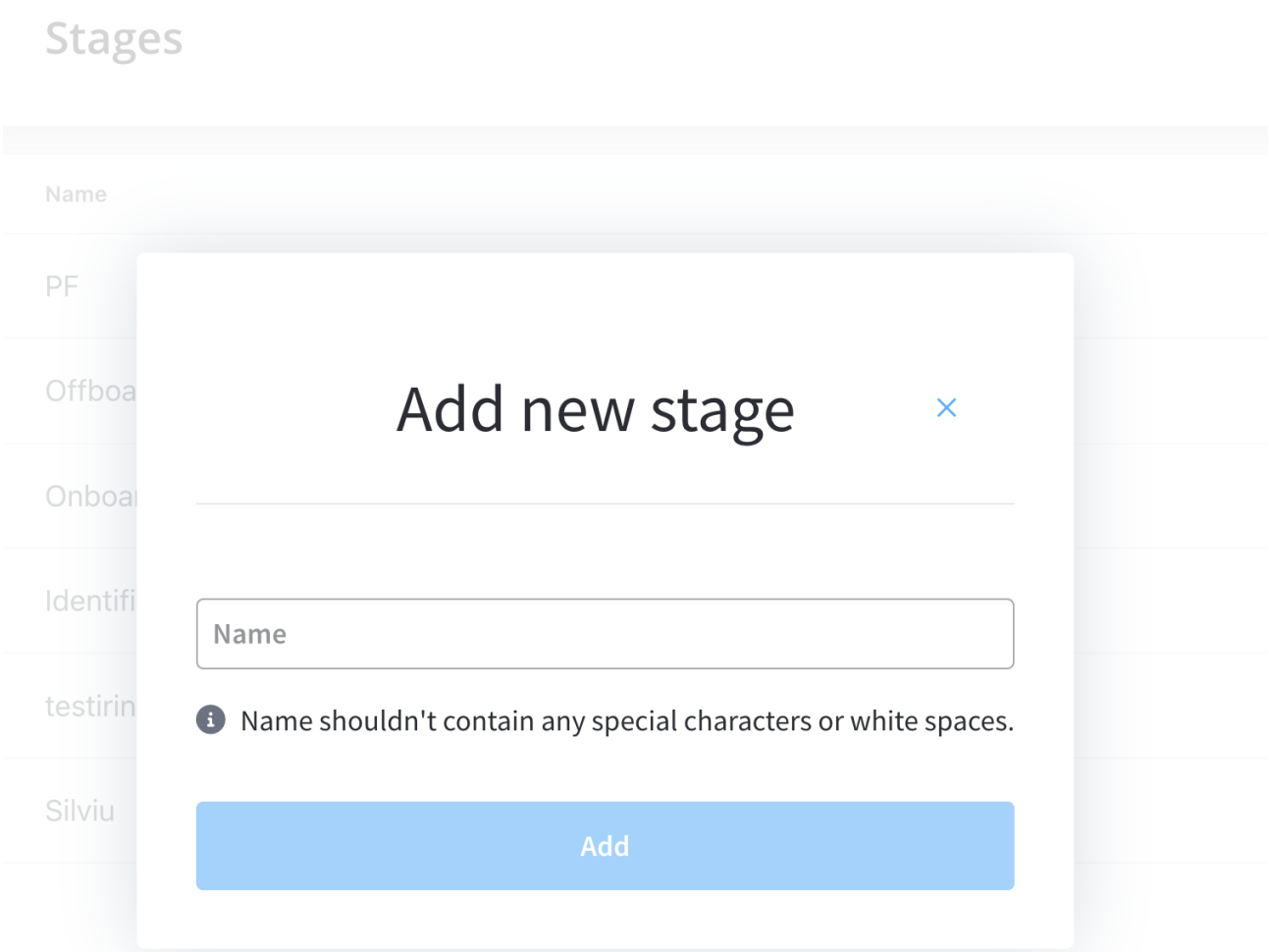
PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Task management / Using stages

You can define specific stages during the execution of a process. Stages are configured on each node and they will be used to trigger an event when passing from one stage to another.

Creating a new stage

To create a new stage, follow the next steps:

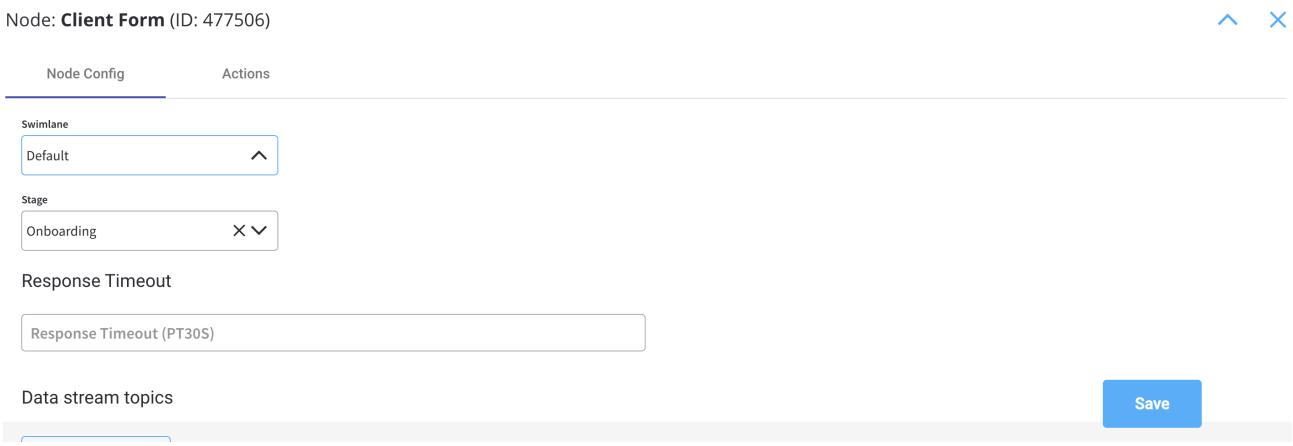
1. Open
   The fallback content to display on prerendering
   .
2. Go to Task Manager and select **Stages**.
3. Click **New Stage.**
4. Fill in the required details.



Assigning a node to a stage

To assign a node to a stage, follow the next steps:

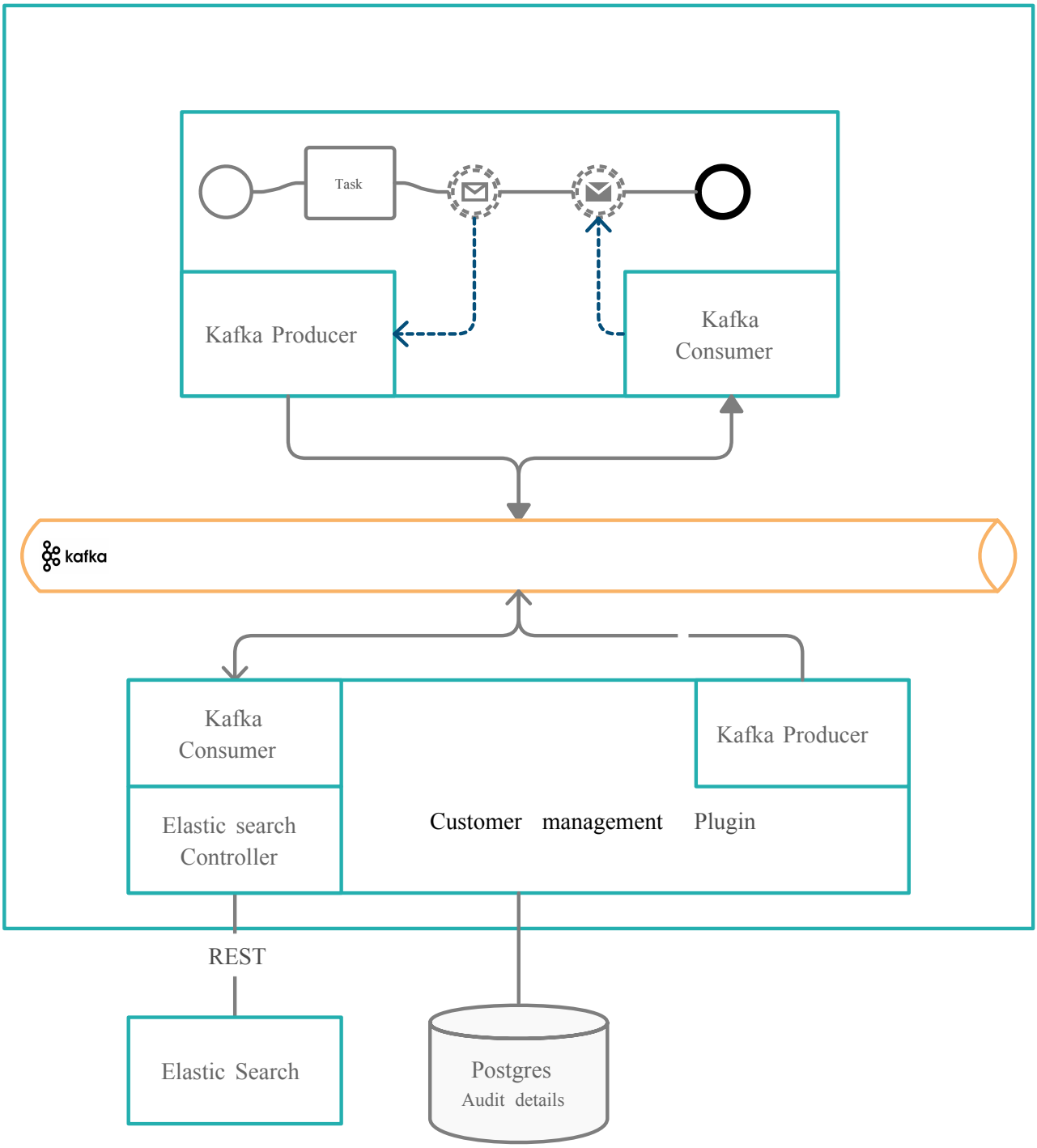
1. Open **FLOWX Designer** and then select your **process**.
2. Choose the node you want to assign and select the **Node Config** tab.
3. Scroll down until you find the **Stage** field and click the dropdown button.
4. Choose the stage you want to assign.



Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Customer management / Using the customer management plugin

The customer management plugin offers the possibility of retrieving customer details from an elasticSearch engine.



The plugin listens for incoming requests on a Kafka topic and sends the reply to the Engine on an outgoing topic.

» Kafka topics for Customer Management

Kafka topics for customer management

Customer Search

> **INFO**
>
> The kafka topics used for the Customer Management plugin can be defined/overwritten using the following environment variables (that can be found in the deployment of the service):
>
> - `KAFKA_TOPIC_CUSTOMER_SEARCH_IN` - used to search customers in the customer management plugin
> - `KAFKA_TOPIC_CUSTOMER_SEARCH_OUT` - used to get the response from the customer management plugin to the Engine.

The request sent to the plugin can use any key that was previously configured in the elasticsearch index where the customers are saved.

Example of an elastic search index:

```
{
  "settings": {
    "analysis": {
      "normalizer": {
        "lowercase_normalizer": {
          "type": "custom",
          "filter": ["lowercase"]
```

```
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "CIF":{
        "type": "keyword",
        "normalizer": "lowercase_normalizer"
      },
      "ClientUniqueIdentifier": {
        "type": "keyword",
        "normalizer": "lowercase_normalizer"
      },
      "CNPFlex":{
        "type": "text"
      },
      "ClientType":{
        "type": "text"
      },
      "ClientCategory":{
        "type": "text"
      },
      "FirstName":{
        "type": "text",
        "fields": {
          "keyword":{
            "type": "keyword",
            "normalizer": "lowercase_normalizer"
          }
        }
      },
      "LastName":{
        "type": "text",
        "fields": {
```

```
          "keyword":{
            "type": "keyword",
            "normalizer": "lowercase_normalizer"
          }
        }
      },
      "CompanyName":{
        "type": "text",
        "fields": {
          "keyword":{
            "type": "keyword",
            "normalizer": "lowercase_normalizer"
          }
        }
      },
      "DateOfBirth":{
        "type": "date",
        "format": "dd.MM.yyyy"
      },
      "IDDocType":{
        "type": "text"
      },
      "IDSeries":{
        "type": "text"
      },
      "IDNumber":{
        "type": "text"
      },
      "IDIssueDate": {
        "type": "date",
        "format": "dd.MM.yyyy"
      },
      "IDExpiryDate":{
        "type": "date",
        "format": "dd.MM.yyyy"
```

```
        },
        "LegalForm":{
          "type": "text"
        },
        "CreatedDatePJ":{
          "type": "date",
          "format": "dd.MM.yyyy"
        },
        "ClientClosedDate": {
          "type": "date",
          "format": "dd.MM.yyyy"
        },
        "LastModifiedDate":{
          "type": "date",
          "format": "dd.MM.yyyy"
        },
        "ListID":{
          "type": "text"
        },
        "MobilePhone":{
          "type": "text"
        }
      }
    }
  }
```

With this index configuration we can search for customers using any key:

Key examples

Example 1 - using only the "ClientUniqueIdentifier" key:

```
{
"ClientUniqueIdentifier": "1900101223344"
}
```

Example 2 - using "FirstName" and "LastName" keys:

```
{
"FirstName": "TestFirstName",
"LastName": "Test Last Name"
}
```

Example 3 - using "FirstName", "DateOfBirth" and "LegalForm" keys:

```
{
"FirstName": "TestFirstName",
"DateOfBirth": "01.01.1990",
"LegalForm": "PF"
}
```

Keys description

- **customers** - list of customers found in the customer management, in the used elasticsearch index, maximum 10 results
- **hasMore** - boolean, true if number of results are bigger than 10, false if the number of results are equal or smaller than 10
- **error** - error description if the request returned an error

Topic name example:

```
ro.flowx.updates.sandbox.customer.management.response
```

Sent body example:


```
"searchResults" : {
    "customers" : [ {
      "id" : "CL12345",
      "firstName" : "John Doe",
      "lastName" : "Doe",
      "birthDate" : "27.02.1982",
      "cui" : "1820227103865_84",
      "companyName" : "",
      "clientCategory" : "PF",
      "clientType" : "PF",
      "idSeries" : "RT",
      "idNumber" : "879948",
      "idDocType" : "CI",
      "idExpiryDate" : "27.02.2023",
      "legalForm" : "",
      "listId" : "4691602",
      "mobilePhone" : "0711111111",
      "attributes" : null,
      "type" : "PF"}],
    "hasMore" : false,
    "error" : null
  }
```


Was this page helpful?

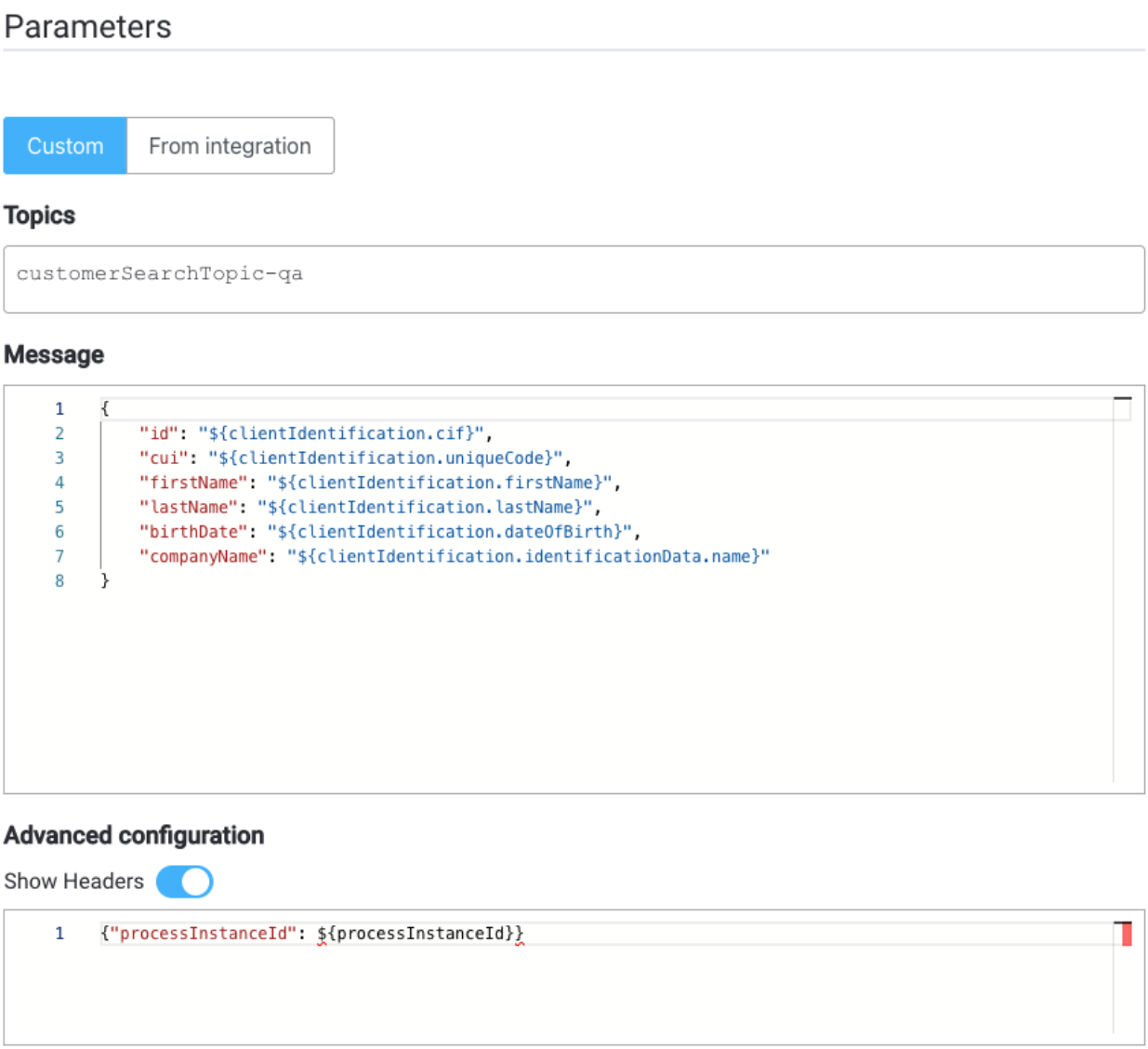
PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 Customer management / Customer management plugin example

Integrate a customer search in a business flow

Follow the next steps to use the user personal number to perform a search query in the customer management plugin.

1. First make sure the details about customers are indexed in the search engine (for example, elasticSearch).
2. Open
   The fallback content to display on prerendering
   web app and create a
   The fallback content to display on prerendering
   .
3. Add a **Kafka send event** node.
4. Configure the **Kafka send event** node by adding the following elements:

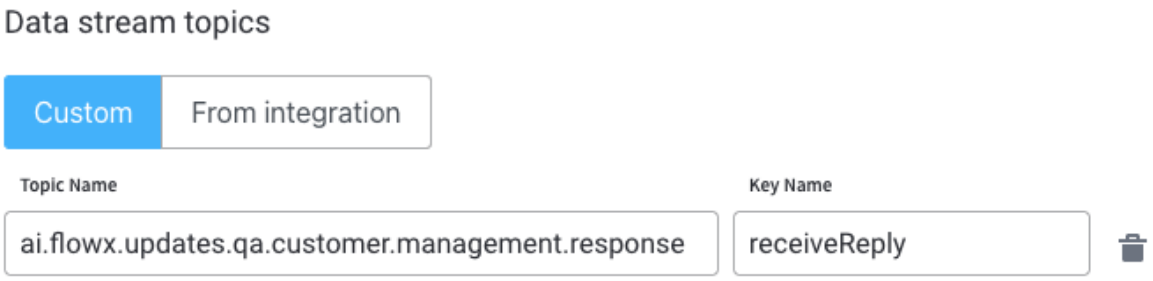
- Kafka topic - defined on the `KAFKA_TOPIC_CUSTOMER_SEARCH_IN` environment variable
- Message body (example of identifiers for an indexed customers):



> **INFO**
>
> For more examples of keys, check **Using the customer management plugin**.

5. Add a **Kafka receive event**.

6. Configure the topic on which you want to receive the response from the CRM, on the value of `KAFKA_TOPIC_CUSTOMER_SEARCH_OUT` environment variable.



Response example:

```
"searchResults" : {
    "customers" : [ {
      "id" : "ID3456",
      "firstName" : "Jane Doe",
      "lastName" : "Doe",
      "birthDate" : "27.02.1980",
      "cui" : "1820227103840_84",
      "companyName" : "",
      "clientCategory" : "PF_INTL",
      "clientType" : "PF",
      "idSeries" : "RT",
      "idNumber" : "879948",
      "idDocType" : "CI",
      "idExpiryDate" : "27.02.2023",
      "legalForm" : "",
      "listId" : "4691602",
      "mobilePhone" : "0711111111",
```

```
      "attributes" : null,
      "type" : "PF"}],
    "hasMore" : false,
    "error" : null
  }
```

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 📦 OCR plugin

The OCR (Optical Character Recognition) plugin is a powerful tool that enables you to read barcodes and extract handwritten signatures from .pdf documents with ease.

Before using the OCR service for reading barcodes and extracting signatures, please note the following requirements:

⚠ CAUTION

- All *.pdf documents that are sent to the OCR service for reading barcodes and extracting handwritten signatures should be scanned at a minimum resolution of 200DPI (approximately 1654x2339 px for A4 pages)
- Barcode is searched on the top 15% of each image (scanned page)
- Signatures are detected on boxes with a border: 4px black solid

- Only two signatures per image (scanned page) are detected.

- All *.pdf documents should be scanned at a minimum resolution of 200DPI (approximately 1654x2339 px for A4 pages).
- The barcode is searched in the top 15% of each scanned page.
- Signatures are detected within boxes with a 4px black solid border.
- The plugin detects up to two signatures per scanned page.
- Only two signatures per image (scanned page) are detected.

INFO

The plugin supports **1D Code 128** barcodes. For more information about this barcode type, please refer to the documentation here.

Using the OCR plugin

You can utilize the OCR plugin to process generic document templates by either using a specific flow on FLOWX.AI (HTML template) or any other document editor.

INFO

Using a specific flow on FLOWX.AI offers several advantages:

- Centralized management of templates and flows within a single application.
- Access to template history and version control.

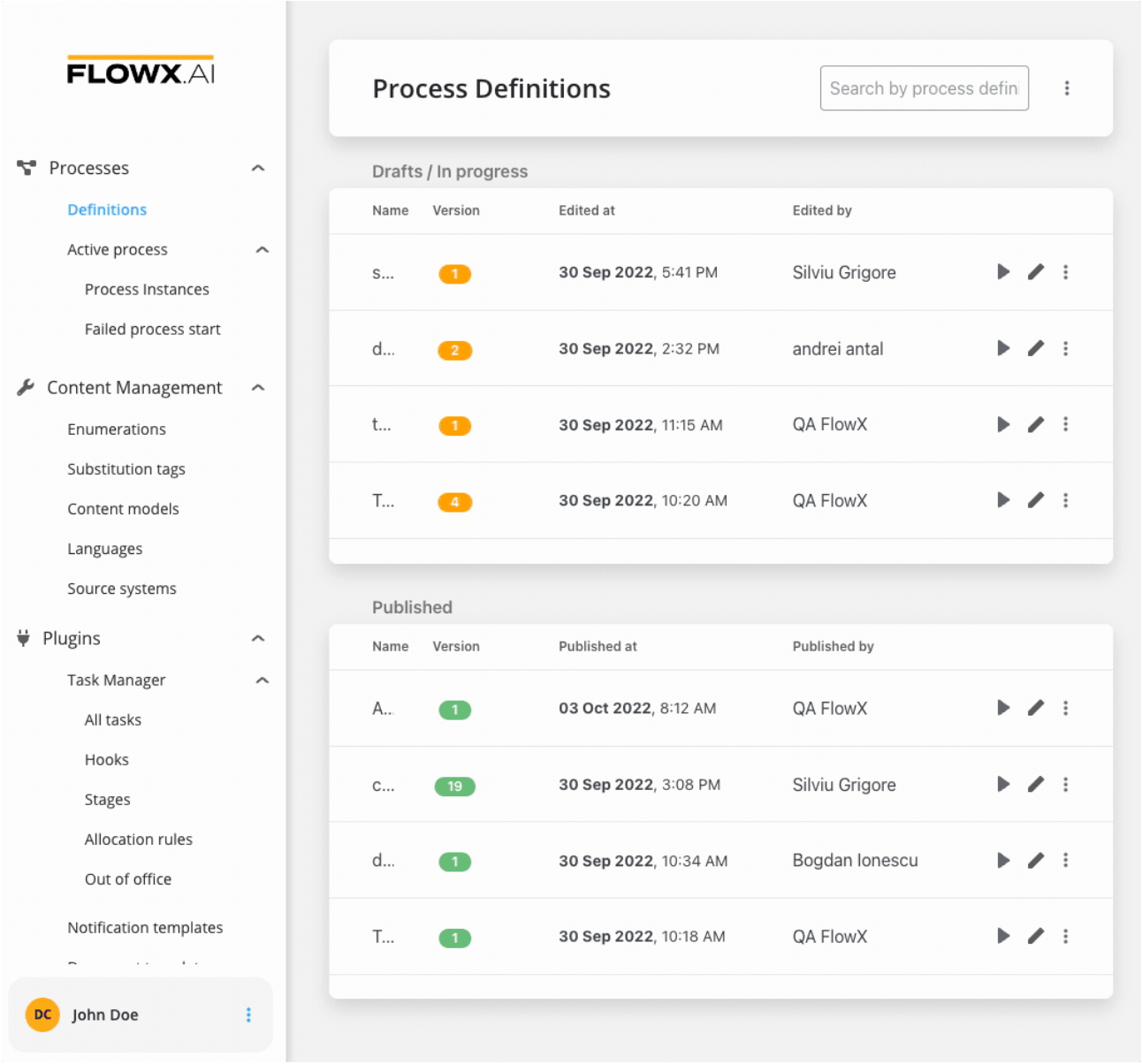
Use case

1. Prepare and print generic document templates.
2. End-users complete, sign, and scan the documents.
3. Upload the scanned documents to the flow.
4. FLOWX validates the template (barcode) and the signatures.

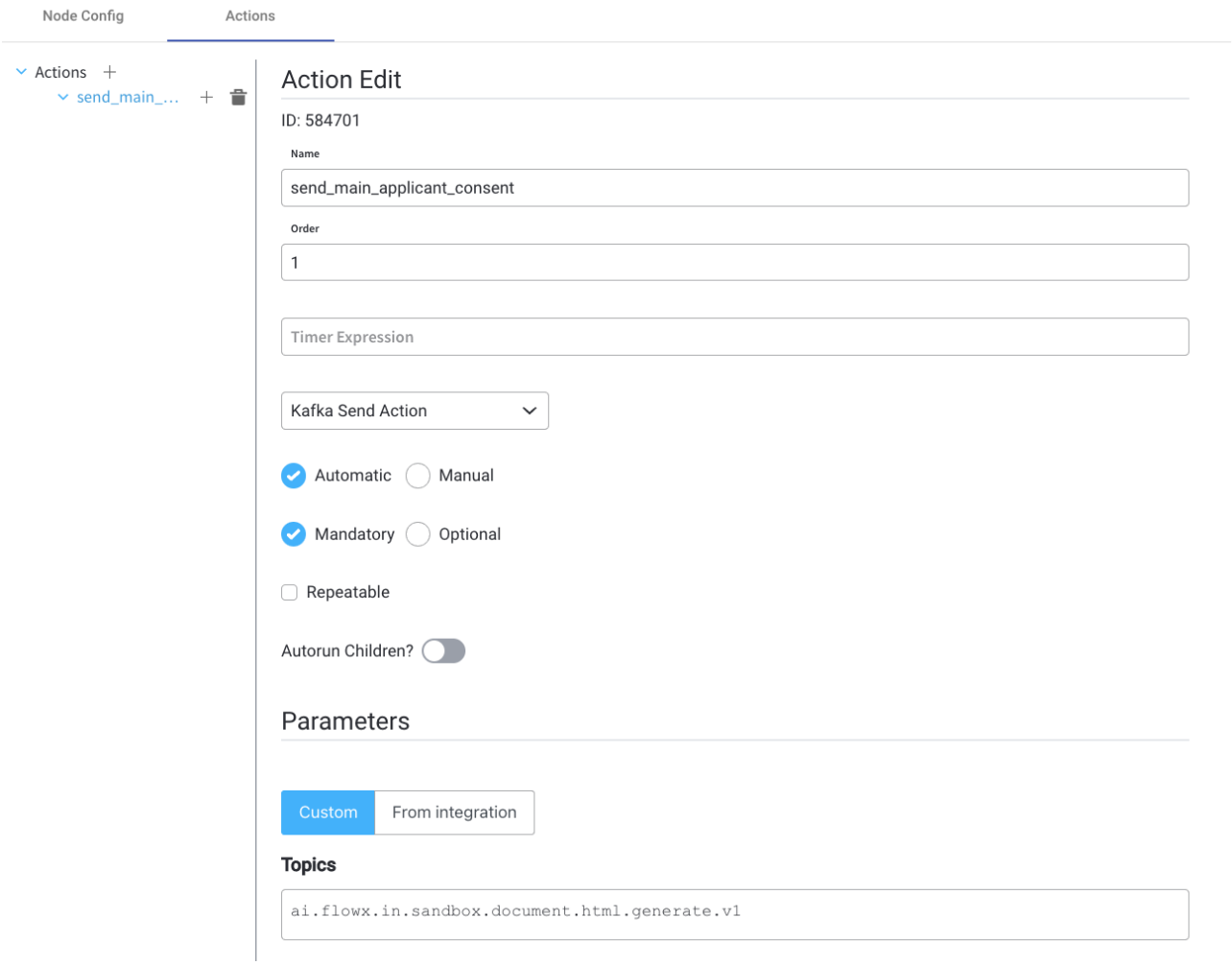
Scenario for FLOWX.AI generated documents

1. Utilize the **Documents plugin** to create a **document template**.

» Generating documents based on templates



2. Create a process and add a **Kafka Send Action** to a **Message event send** node. Here you specify the **kafka topic** (address) where the template will be generated.

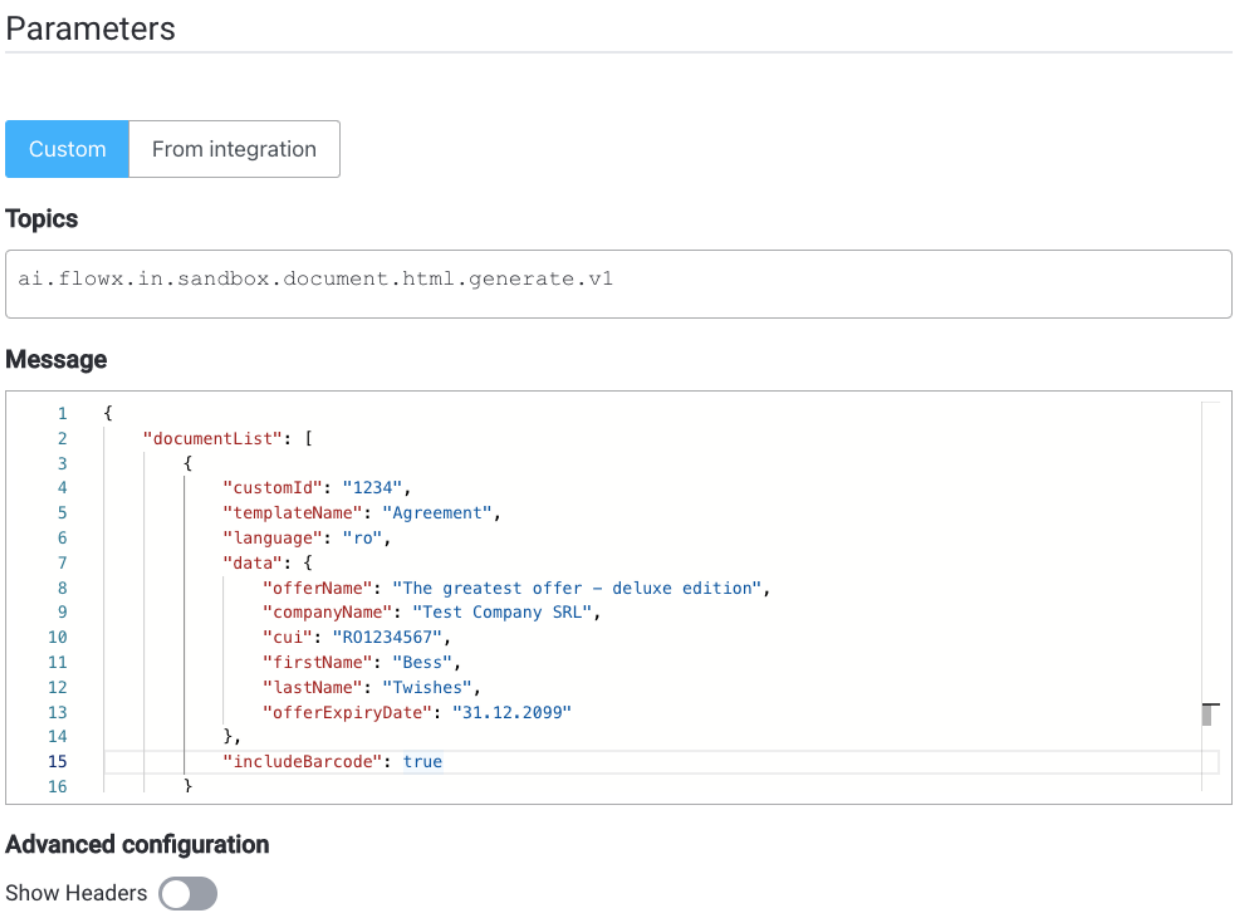


> **INFO**
>
> The Kafka topic for generating the template must match the topic defined in the `KAFKA_TOPIC_DOCUMENT_GENERATE_HTML_IN` variable. Refer to the **Kafka configuration guide** for more details. For additional information, please see the **Documents plugin setup guide**.

3. Fill in the **Message**. The request body should include the following values:

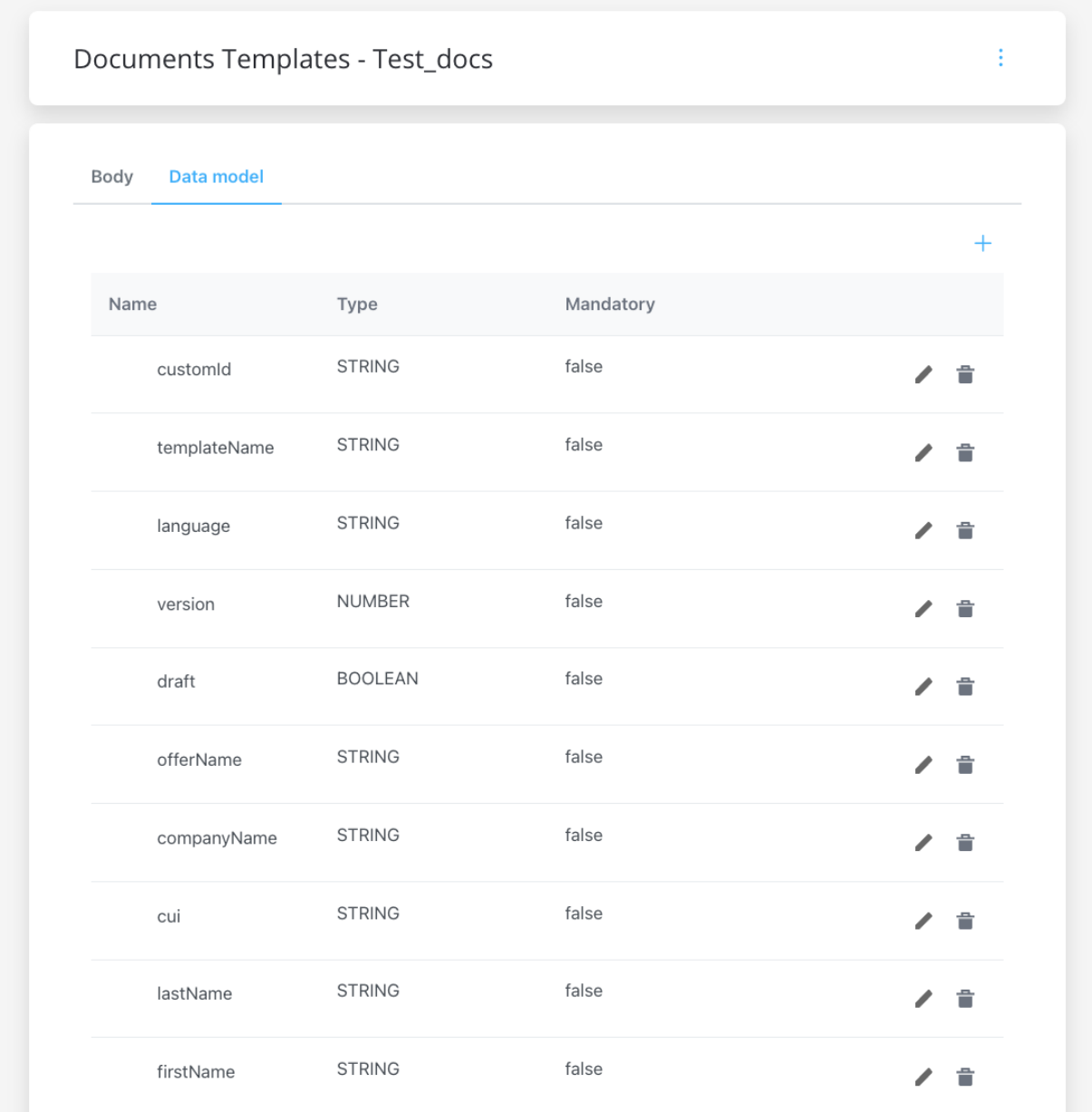
- **documentList** - a list of documents to be generated, including properties such as name and values to be replaced in the document templates

- **customId** - client ID
- **templateName** - the name of the template to be used
- **language**
- **includeBarcode** - true/false
- **data** - a map containing the values that should replace the placeholders in the document template, the keys used in the map should match those defined in the HTML template



INFO

The `data` **parameters** must be defined in the document template beforehand. For more information, check the **WYSIWYG Editor** section.



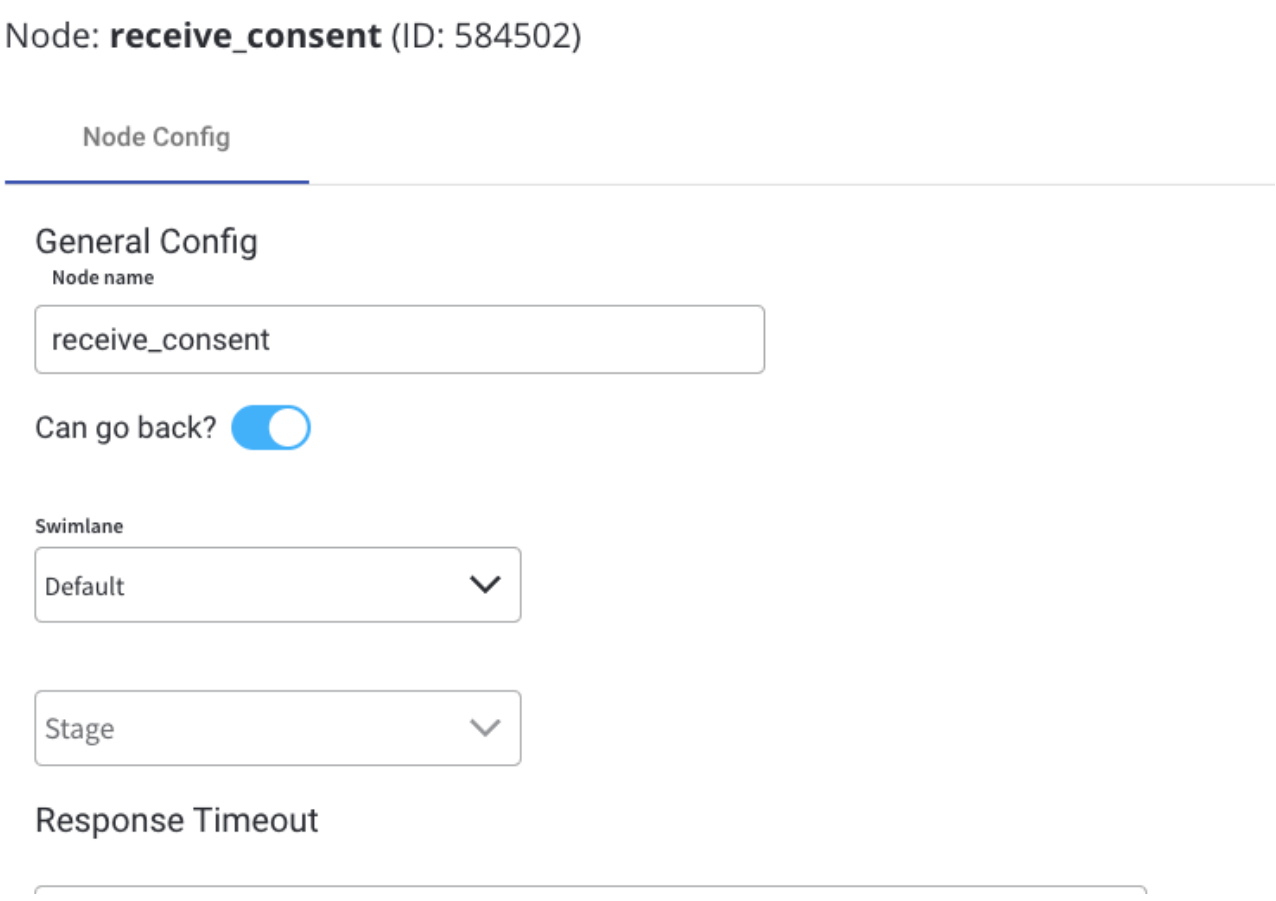
4. Add a **barcode**.

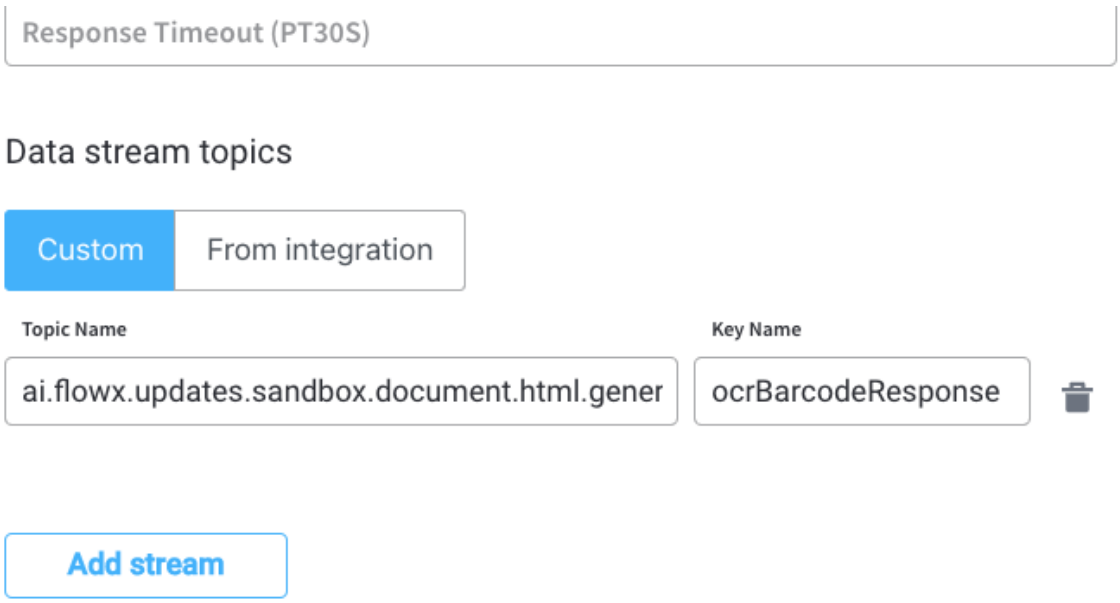
INFO

- to include a **default barcode**, add the following parameter to the message body: `includeBarCode: true`.
- to include a **custom barcode**, set `includeBarCode: false` and provide the desired data in the `data` field

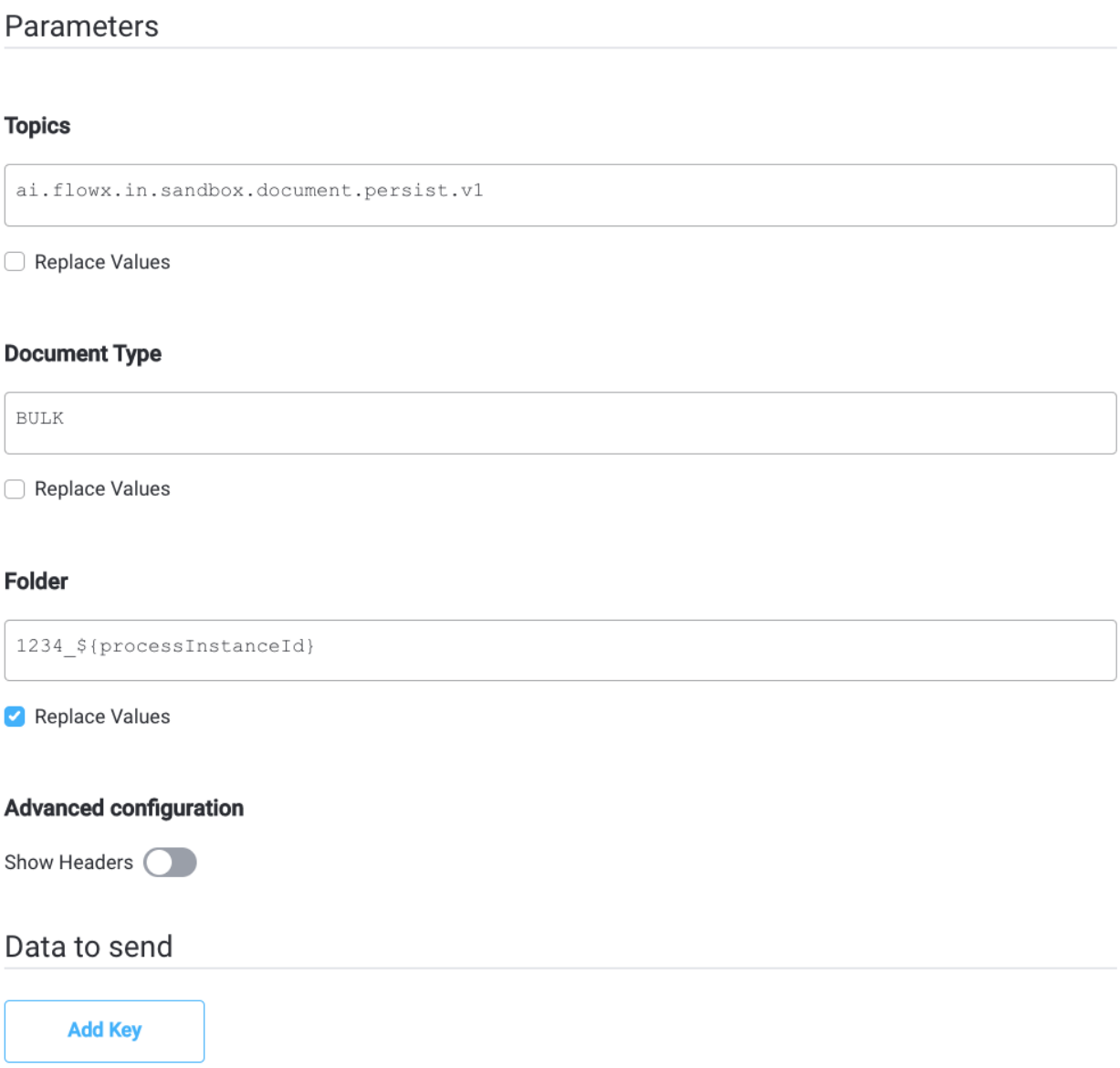
5. Add a **message received event** node and specify the topic where you want to receive the response.

> **CAUTION**
>
> Ensure that the topic matches the one defined in the `KAFKA_TOPIC_DOCUMENT_GENERATE_HTML_OUT` variable.





6. Add a **user task node** and configure an **Upload file action** to send the file (defined by the `KAFKA_TOPIC_DOCUMENT_PERSIST_IN` variable) to the storage solution (for example, S3).



7. Next, the response will be sent back to the kafka topic defined by `KAFKA_TOPIC_DOCUMENT_PERSIST_OUT` environment variable through a callback action/subprocess.
8. Next, send the response to the OCR Kafka topic defined at `KAFKA_TOPIC_OCR_IN` variable (representing the path to the S3 file)

9. Display the result of the OCR validation on the kafka topic defined at `KAFKA_TOPIC_OCR_OUT`.

Setup guide

Refer to the OCR plugin setup guide for detailed instructions on setting up the OCR plugin:

» OCR plugin setup

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / 🗄️ Reporting / Authorization & access roles

IAM solution

> **INFO**
>
> Superset is using be default flask-openid, as implemented in flask-security.

Superset can be integrated Keycloak, an open-source identity and access management solution. This integration enables users to manage authentication and authorization for their Superset dashboards.

» Configuring an IAM solution

Prerequisites

- Keycloak server
  - Keycloak Realm
  - Keycloak Client & broker configured with OIDC protocols
  - client_secret.json
- admin username & password of postgres instance
- Superset Database created in postgresql
- optionally Cert-manager if you want to have SSL certificates on hostnames.

» Superset + Keycloak configuration

Was this page helpful?

PLATFORM DEEP DIVE / Plugins / 📦 Custom Plugins / WYSIWYG editor

FLOWX.AI Designer's WYSIWYG (**"What You See Is What You Get"**) editor enables you to create and modify notification and document templates without the need for complicated coding from the developers. WYSIWYG editors make the creation/editing of any type of document or notification easier for the end-user.

Displaying how the document will be published or printed on the screen, the user can adjust the text, graphics, photos, or other document/notification elements before generating the final output.

WYSIWYG Components

Header

The formatting head of the editor allows users to manipulate/format the content of the document.

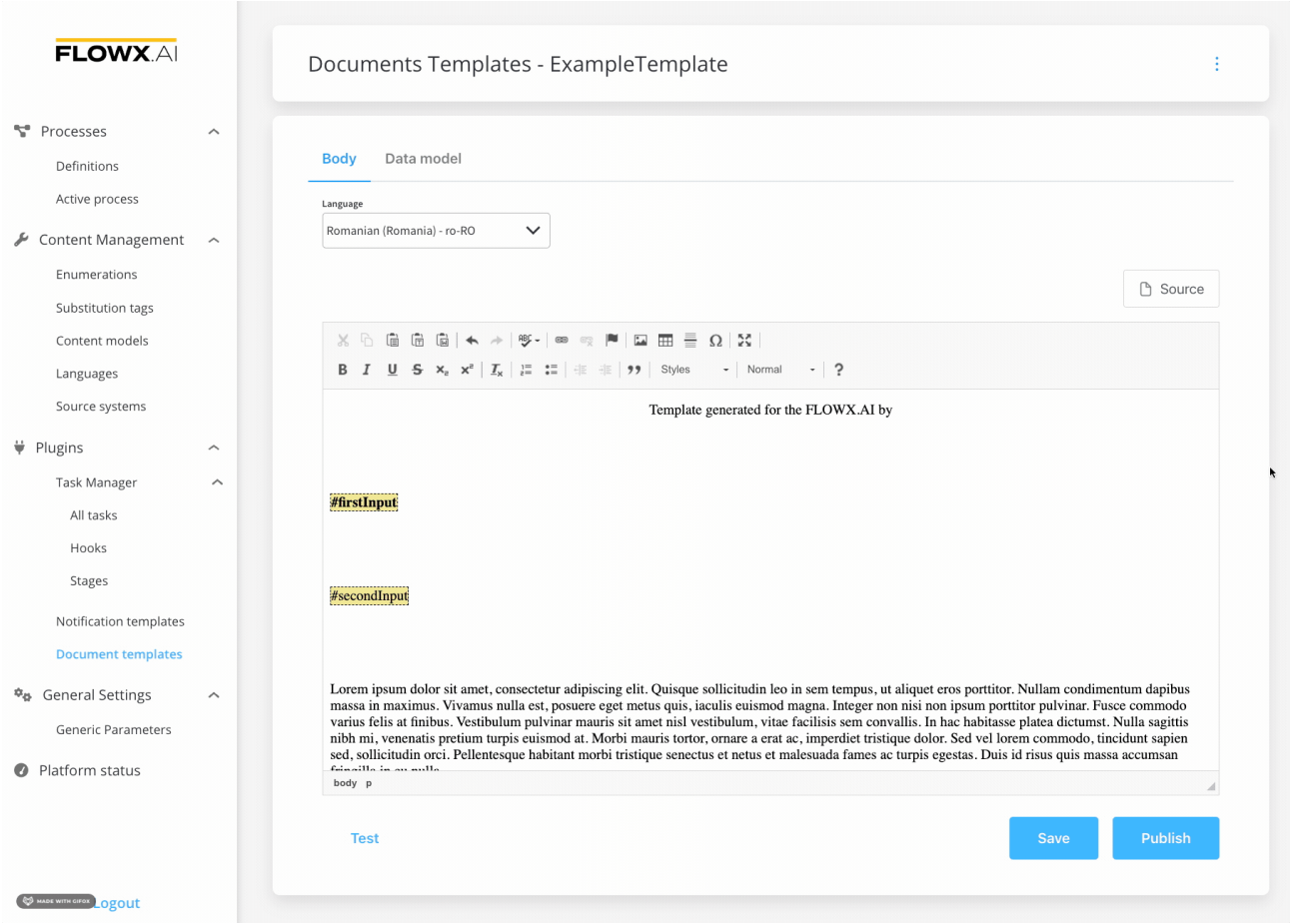
Body

The Body is the main part of the editor where you can edit your template.

> **INFO**
>
> After you defined some parameters in the **Data Model** tab, you can type **"#"** in the body to trigger a dropdown where you can choose which one you want to use.

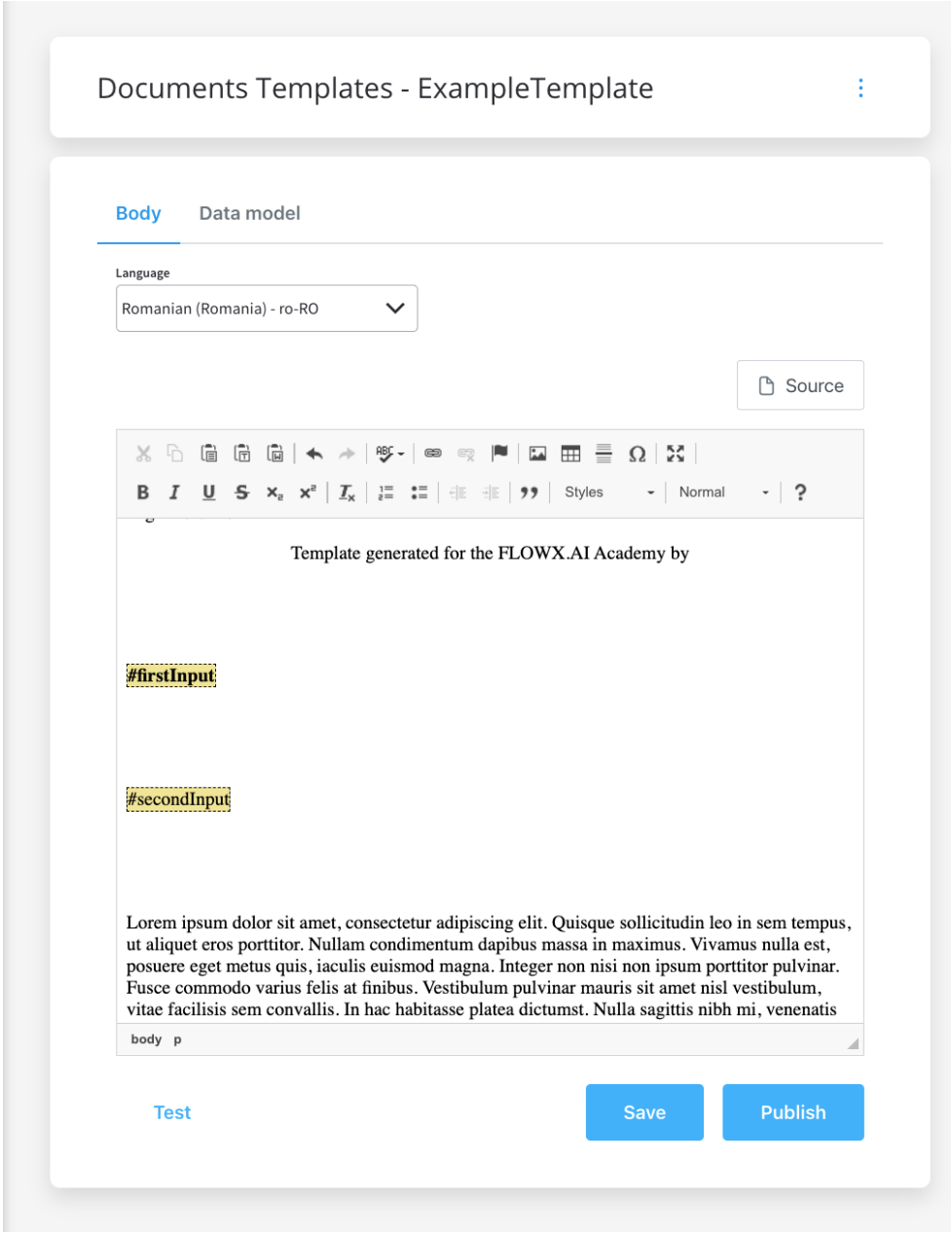
Source

The **Source** button can be used to switch to the HTML editor. You can use the HTML view/editor as a debugging tool, or you can edit the template directly by writing code here.



Document Templates

One of the main features of the document management plugin is the ability to generate new documents based on custom templates and prefilled with data related to the current process instance.

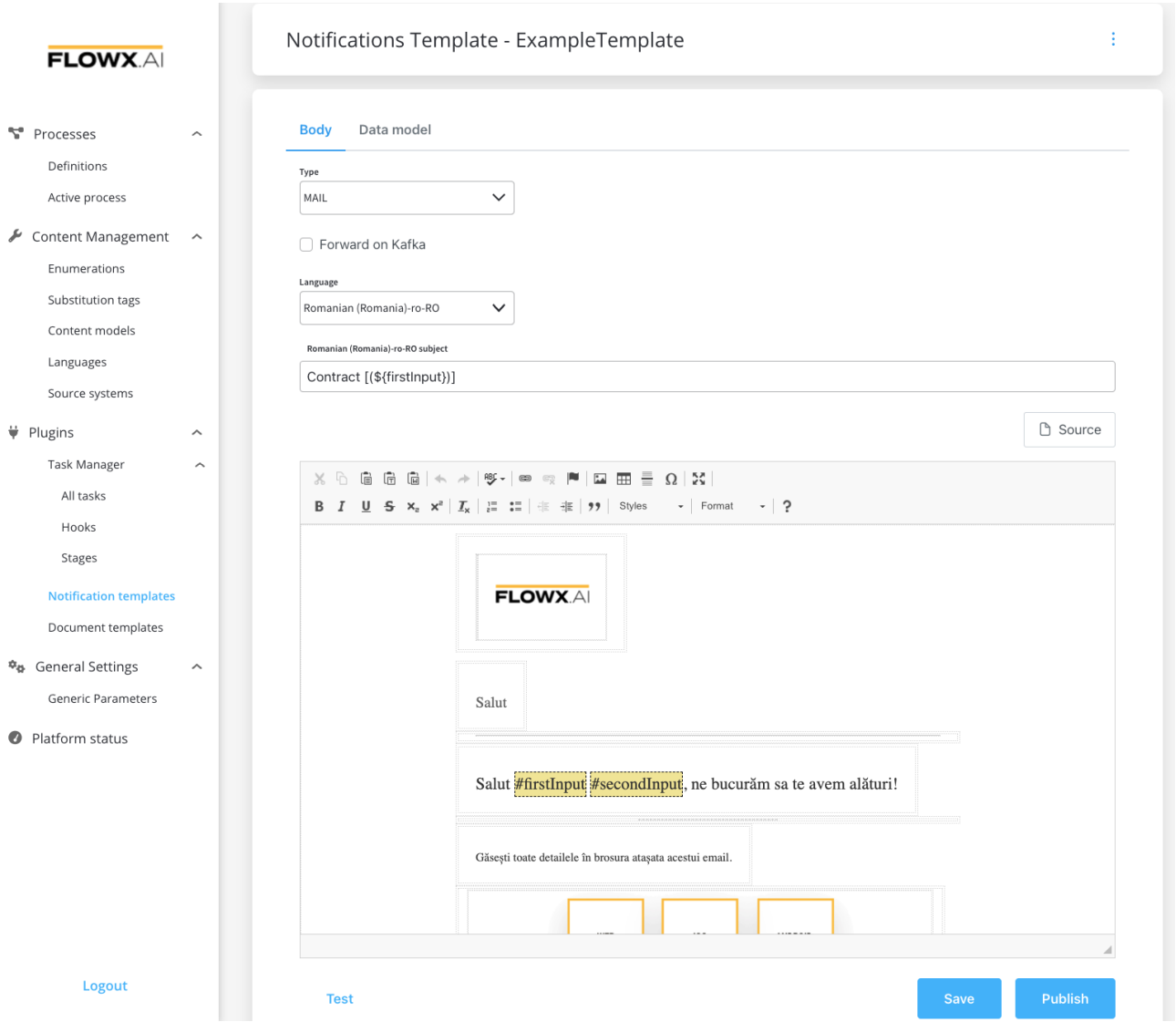


» Documents plugin

Notification Templates

Notification WYSIWYG body has some additional fields (other than documents template):

- **Type** - that could be either MAIL or SMS (SMS, only if there is an external adapter)
- **Forward on Kafka** - if this box is checked, the notification is not being sent directly by the plugin to the destination, but forwarded to another adapter



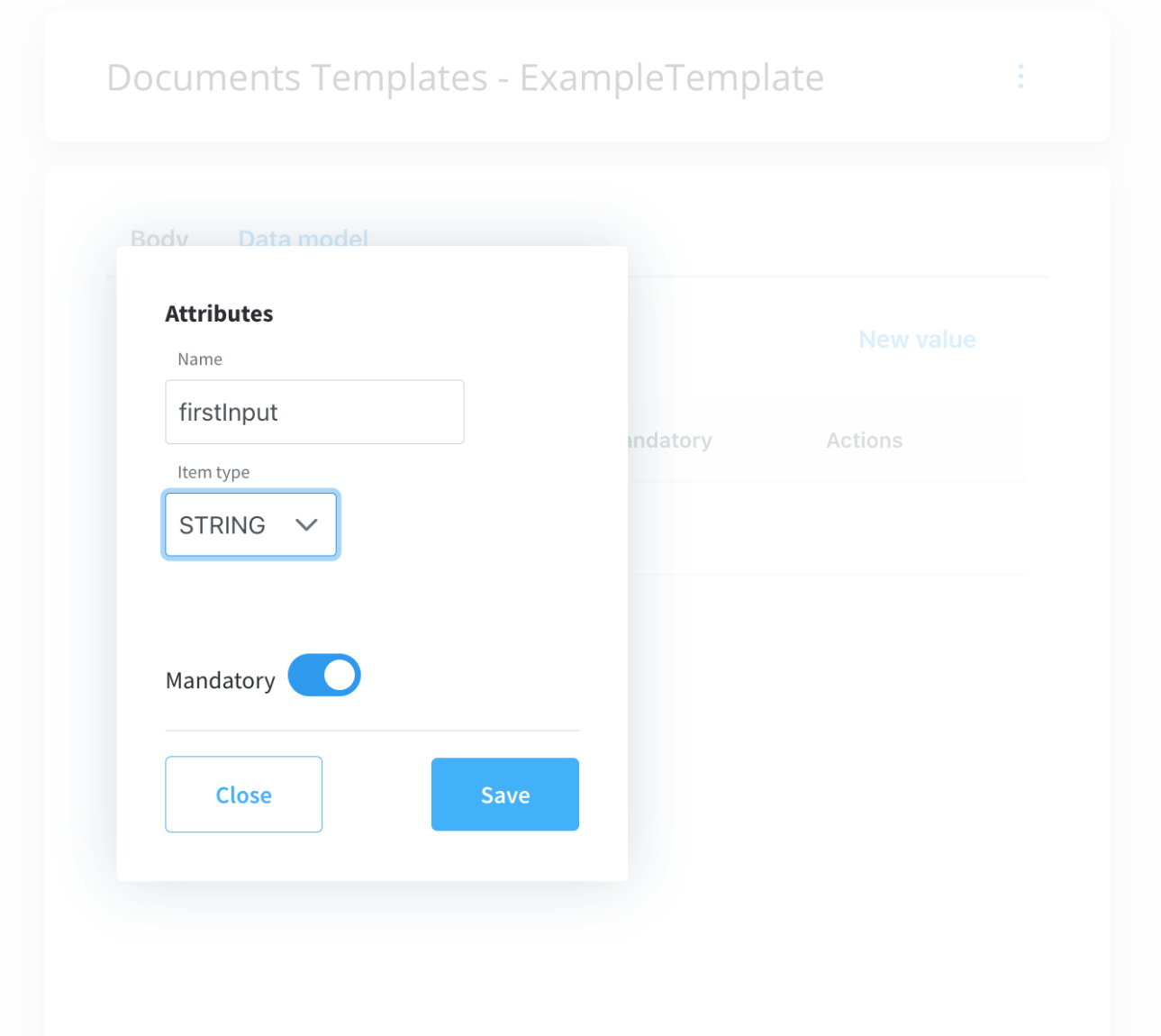
» Managing notifications templates

Data Model

Data Model

Using the data model, you can define key pair values (parameters) that will be displayed and reused in the body. Multiple parameters can be added:

- STRING
- NUMBER
- BOOLEAN
- OBJECT
- ARRAY (which has an additional `item` field)



INFO

Parameters can be defined as mandatory or not. When you try to generate a template without filling in all the mandatory parameters, the following error message will be displayed: "*Provided data cannot be empty if there are any required properties defined.*"

Was this page helpful?