FLOWX.AI

PLATFORM DEEP DIVE / Plugins / Custom Plugins / Documents plugin

Contents

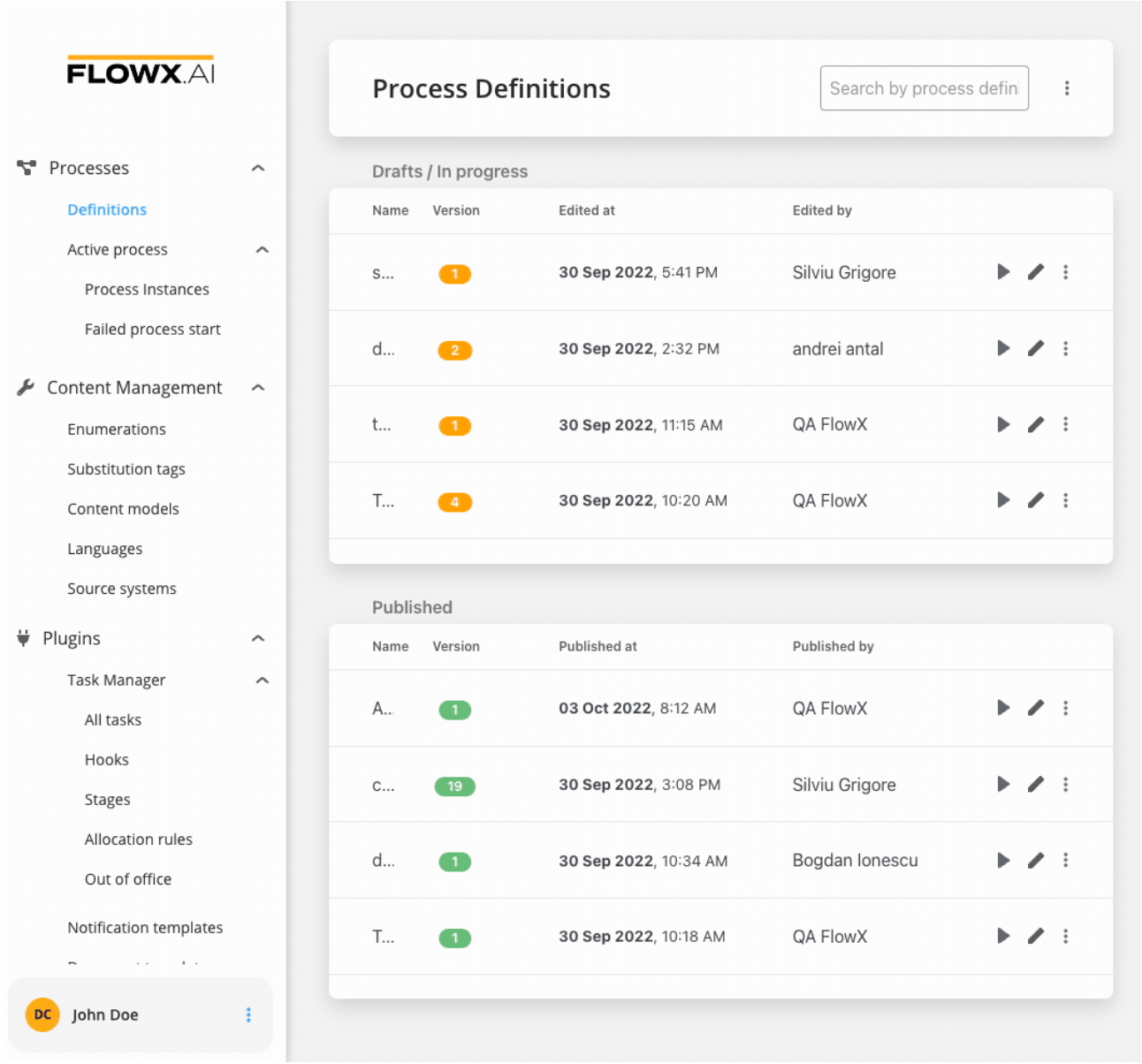
PLATFORM DEEP DIVE / Plugins / Custom Plugins / Documents plugin / Using the plugin / Generating docs

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

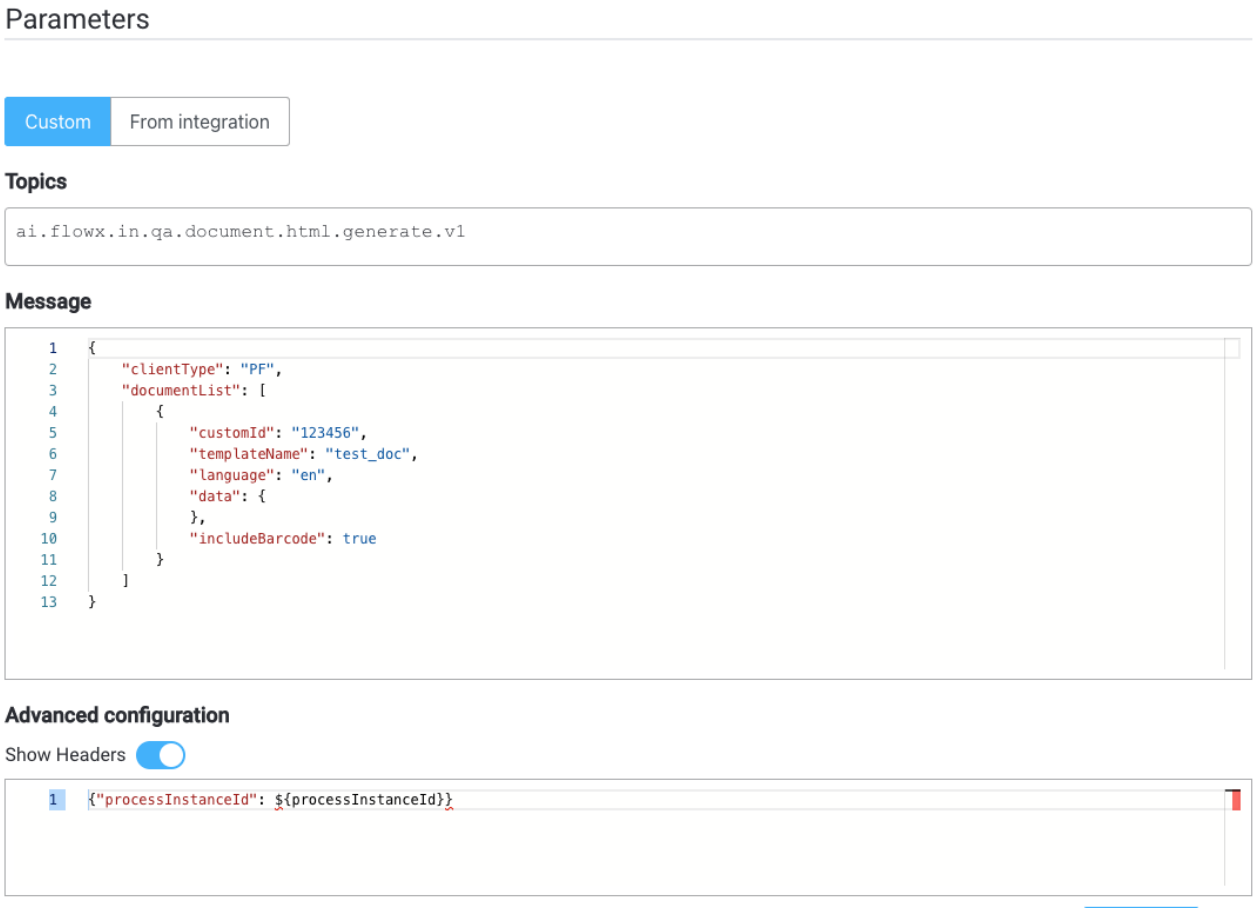
Use the WYSIWYG editor to create a document template.



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

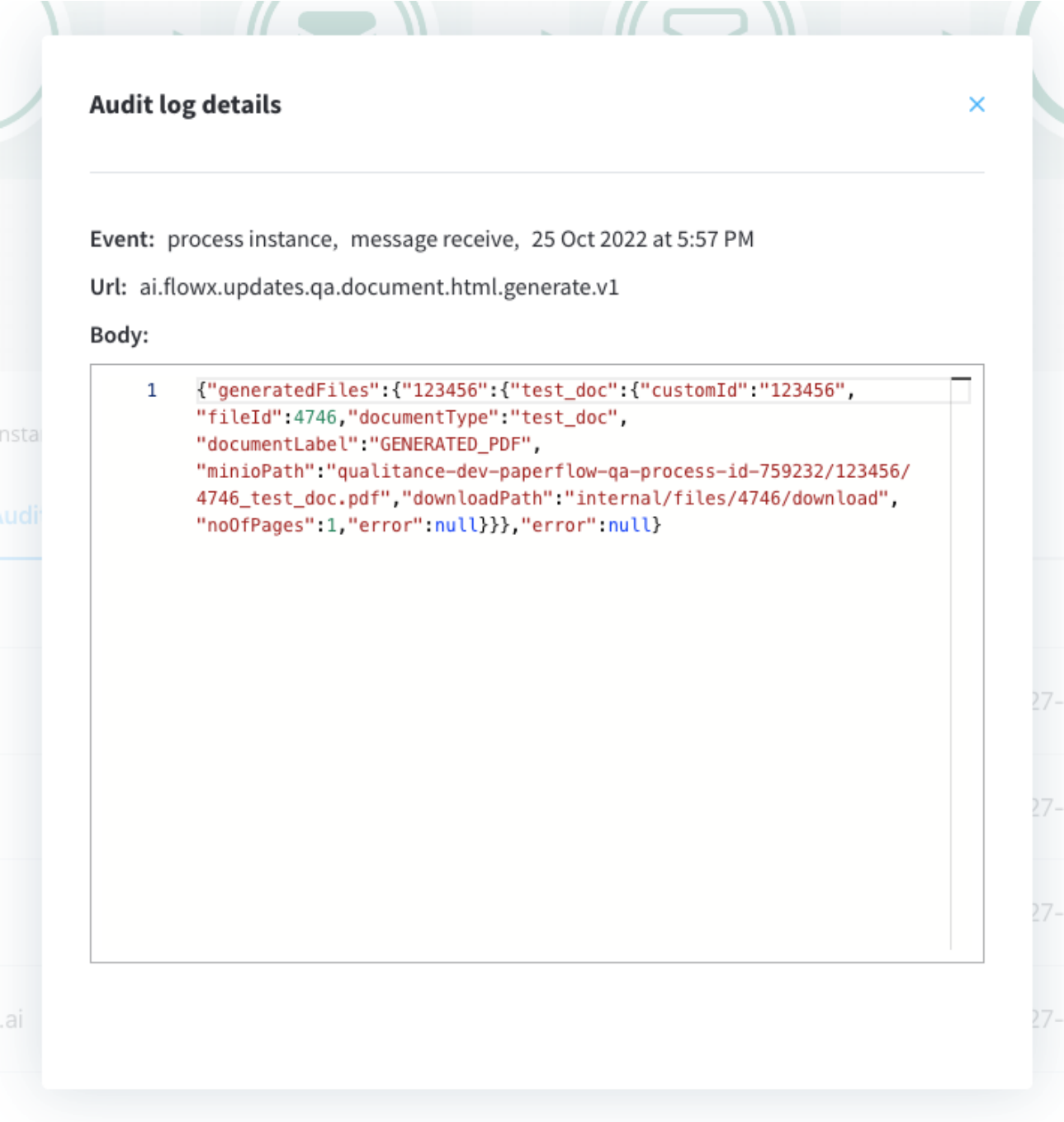
The engine listens for messages on topics with specific naming patterns. Make sure to use an outgoing topic name that matches the pattern configured in the engine.

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

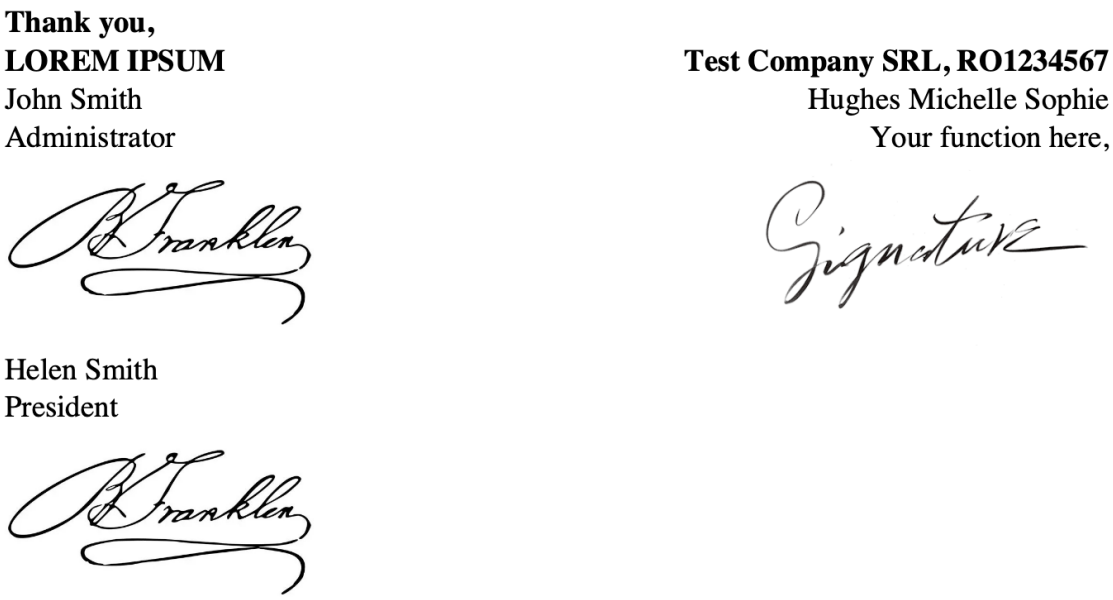
```
<span th:if="${pjCLient==true}">
    <p><b>PJ section, visible only if pjCLient = true</b>
</p>
    <p><span th:text="${termTechnicalServices}"></span></p>
</span>
<span th:if="${pjCLient==false}">
    <p><b>PF section, visible only if pjCLient = false</b>
</p>
    <p><span th:text="${termInsuranceServices}">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

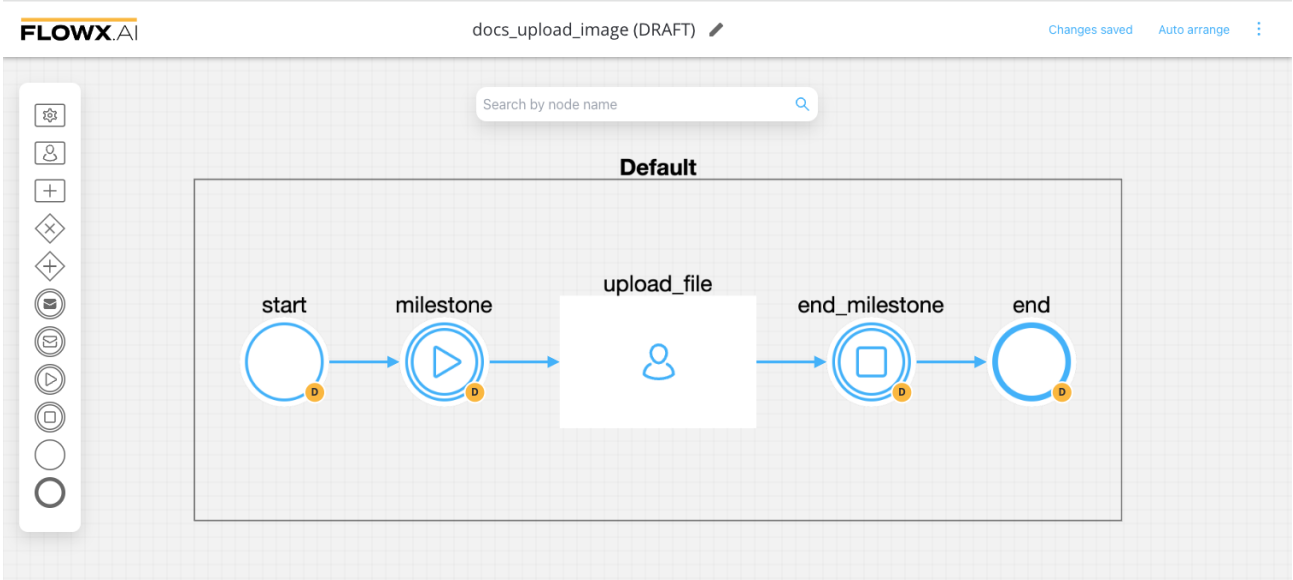
Was this page helpful?

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

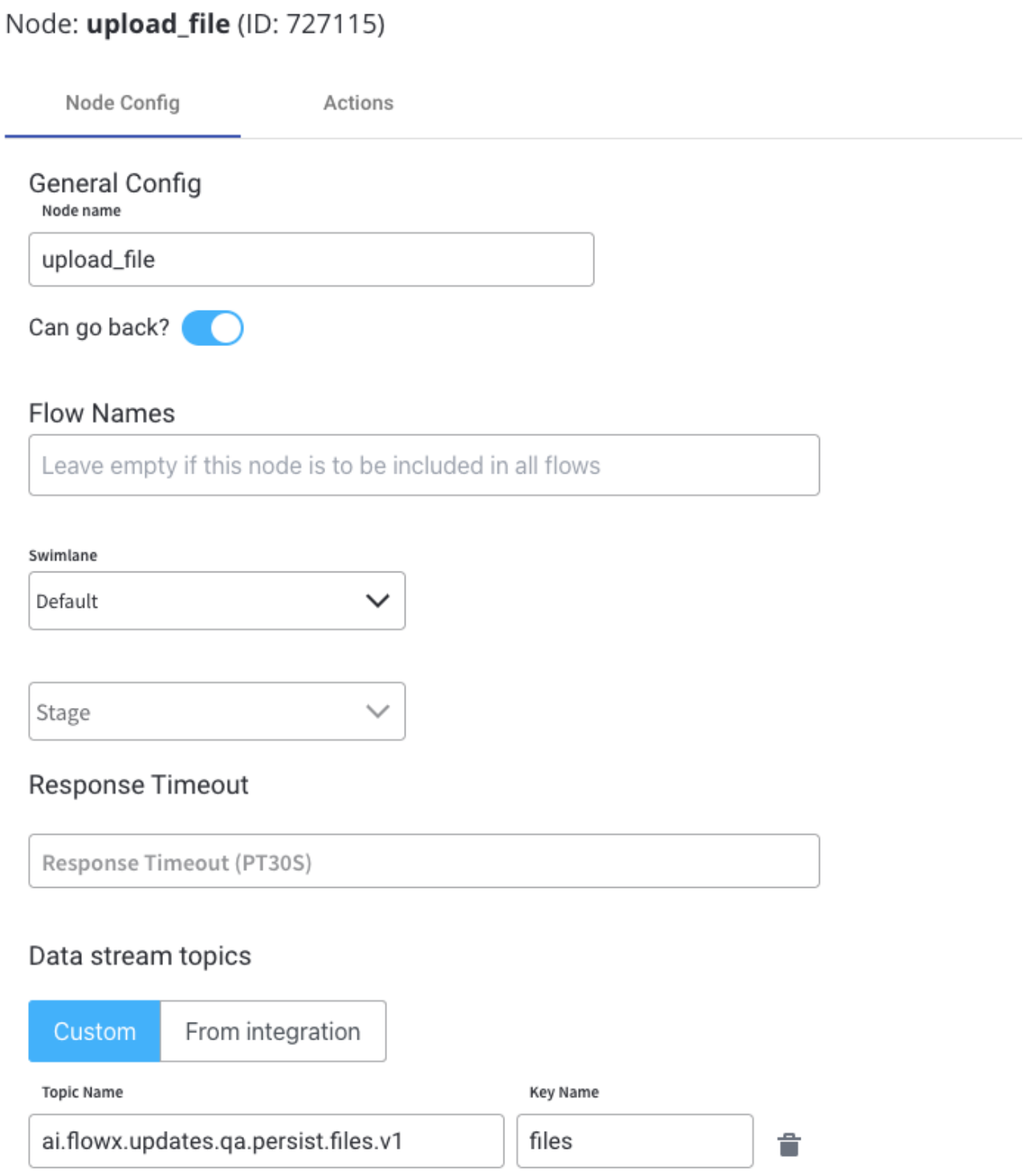
User task node

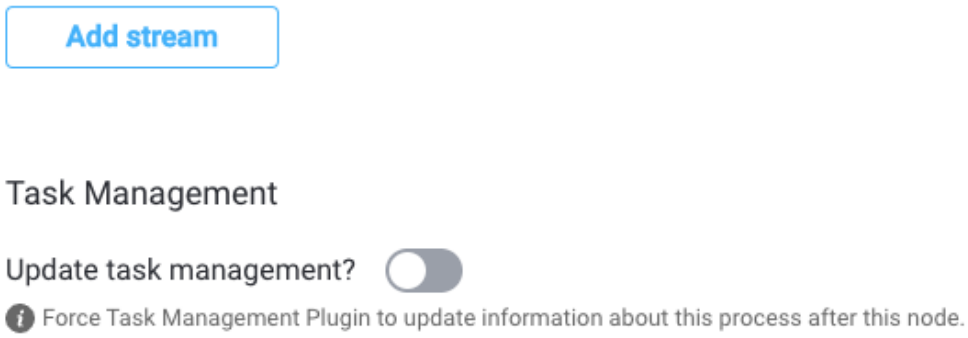
Node Config

- **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.

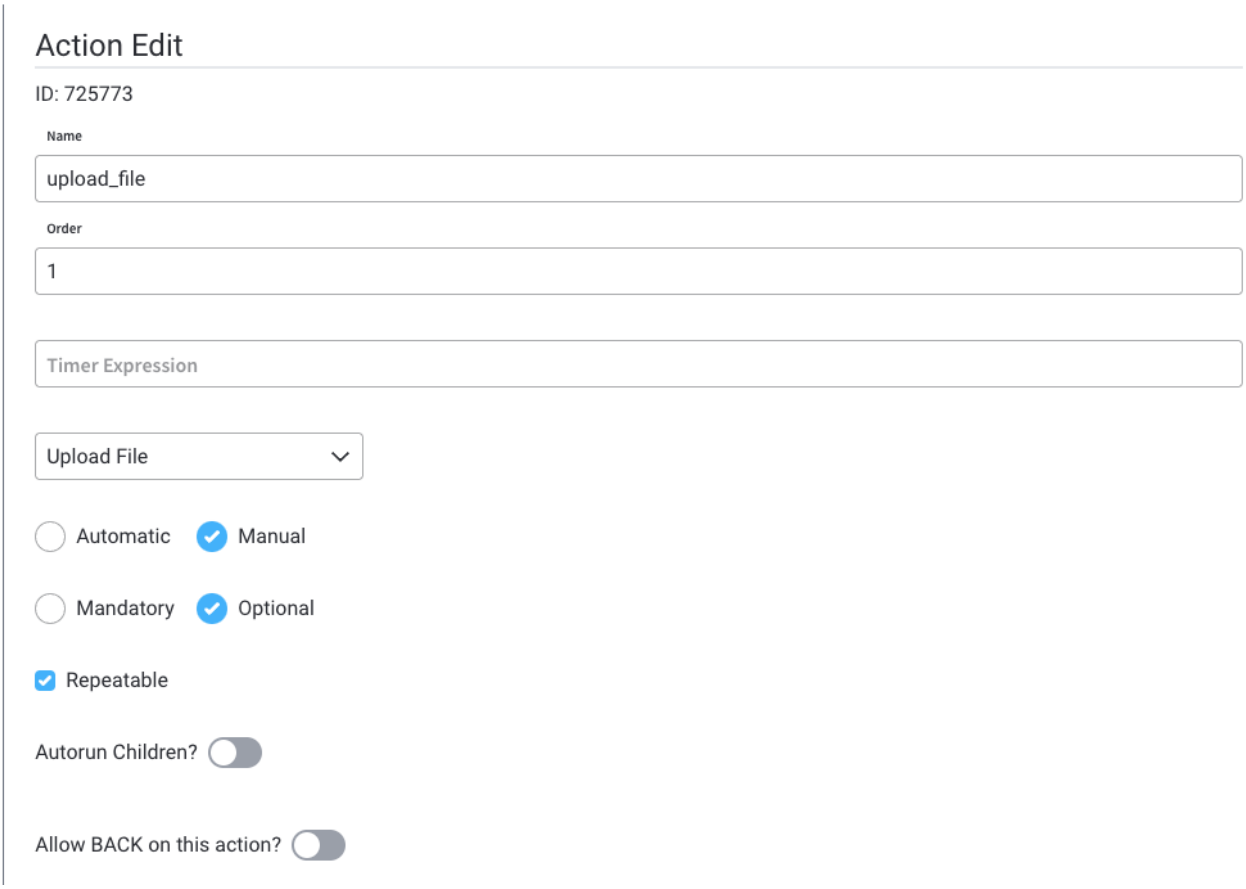




Actions

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

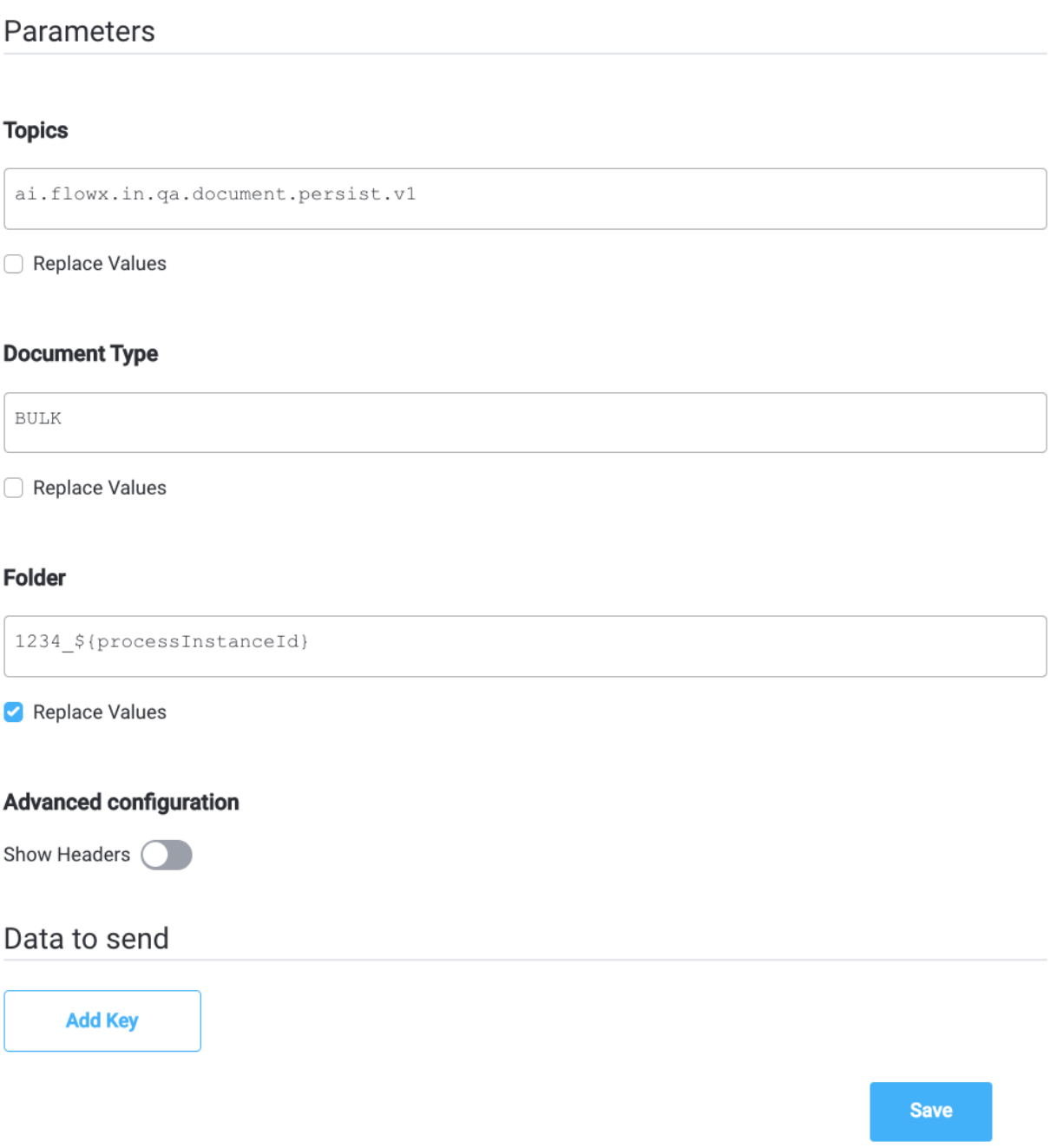
- **Topics**: Set it to `ai.flowx.in.document.persist.v1`, extracted from `KAFKA_TOPIC_DOCUMENT_PERSIST_IN`.
- **Document Type**: Set it to BULK.
- **Folder**: Allows you to configure a value by which the file will be identified in the future.
- **Advanced Configuration (Show Headers)**: Represents a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

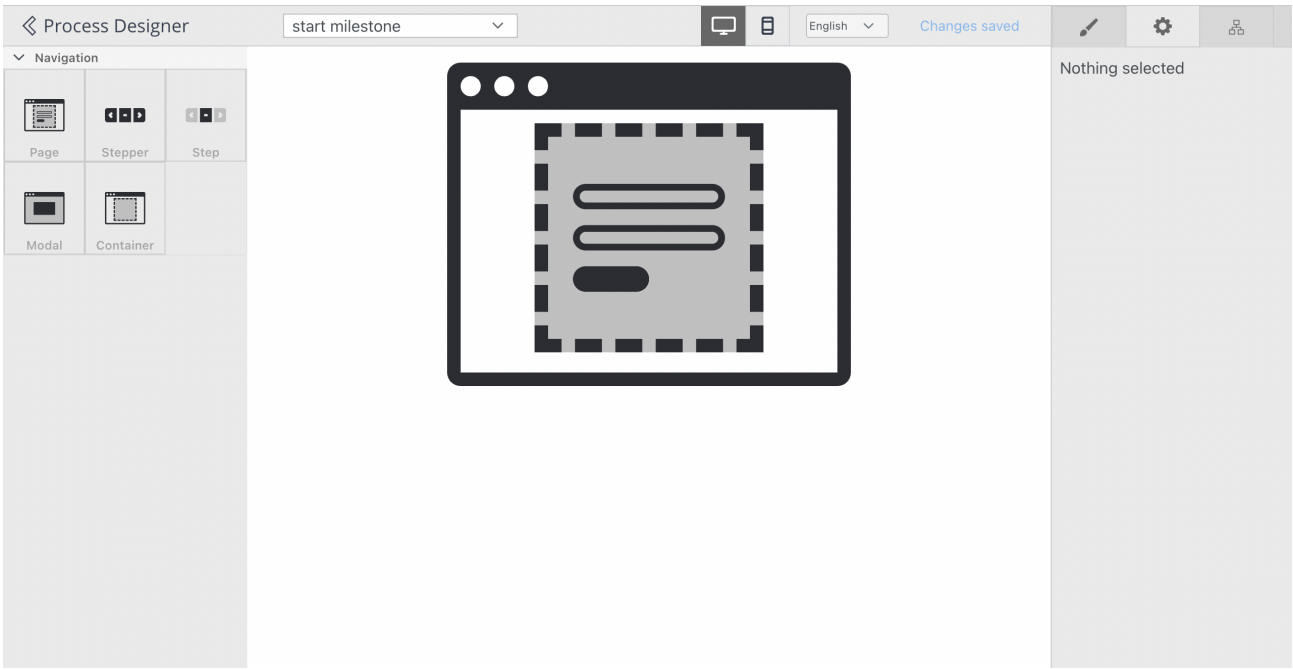
The above examples of topics are extracted from an internal testing environment. When setting topics for other environments, follow this pattern: `ai.flowx.updates.{{environment}}.document.persist.v1`.



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.

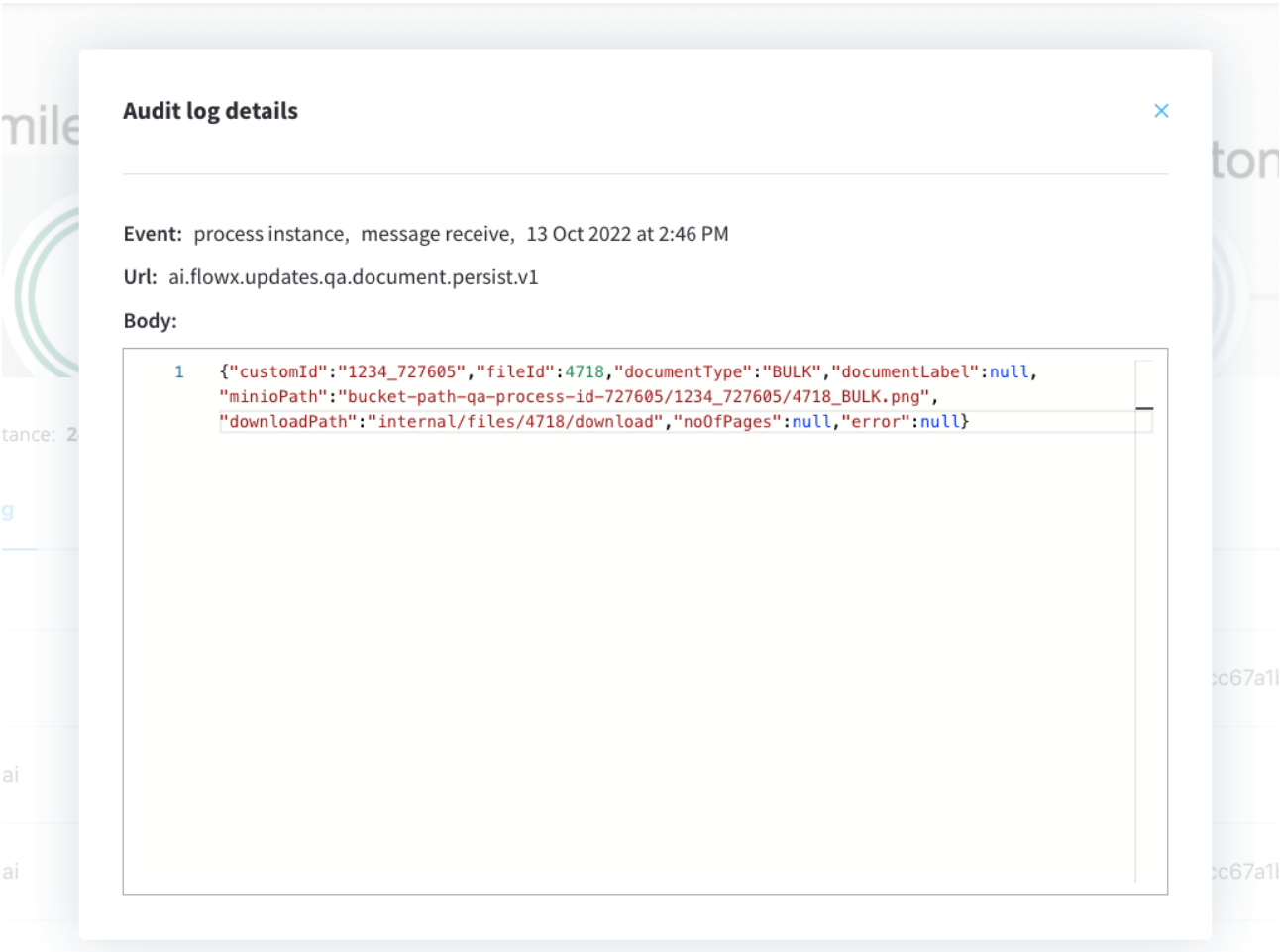


Receiving the reply

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

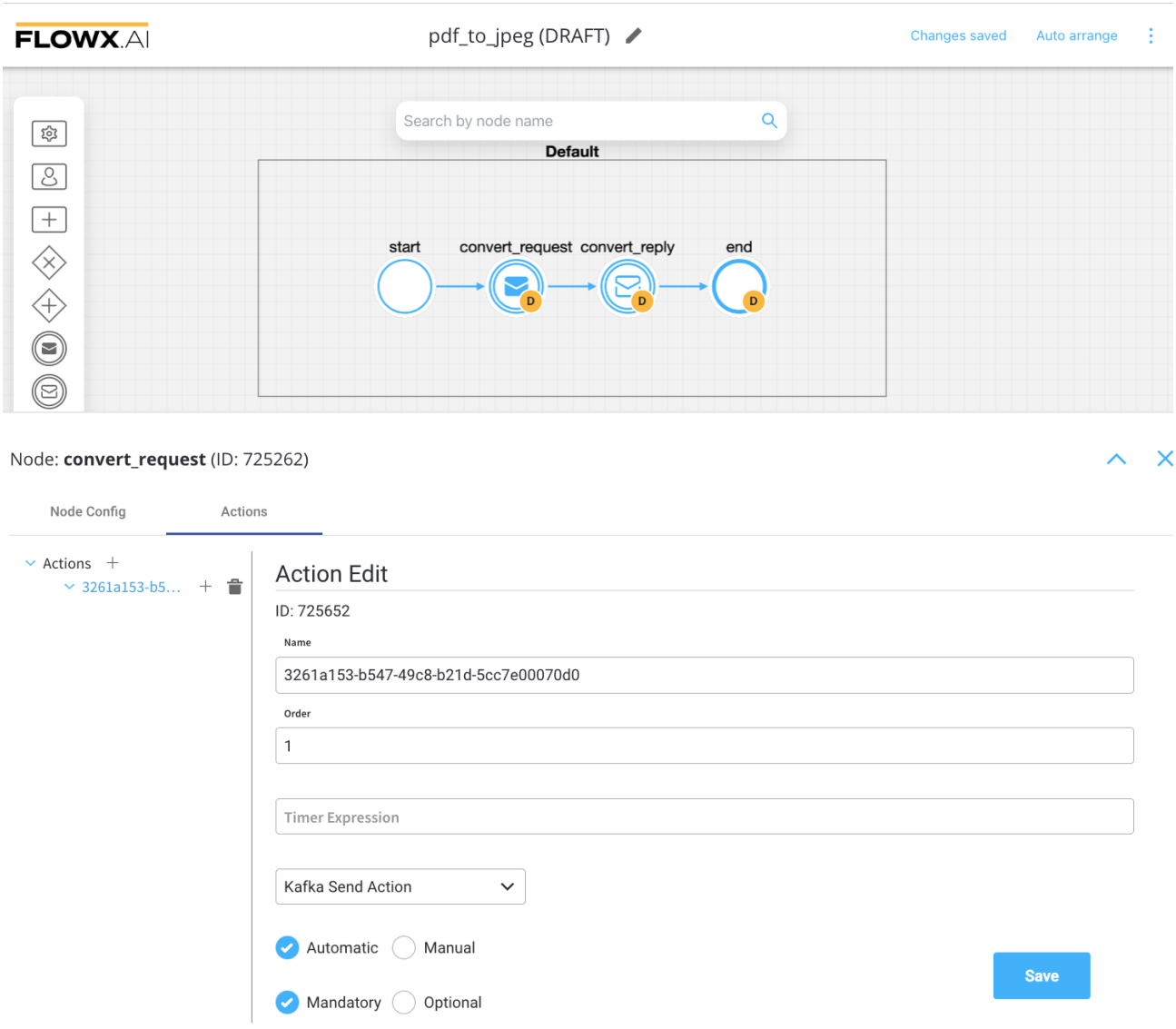
> **CAUTION**
>
> Currently, the supported conversion method is from **PDF** to **JPEG**.

Sending the request

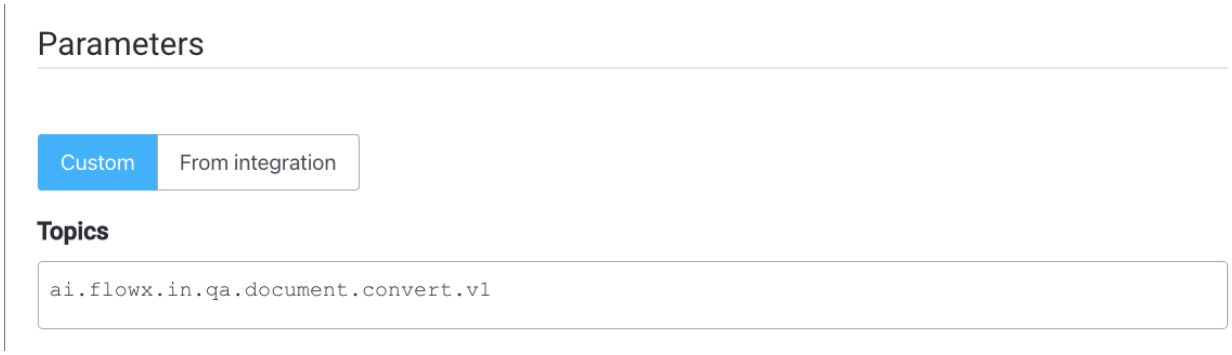
To create a process that converts a document from PDF to JPEG format, follow these steps:

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

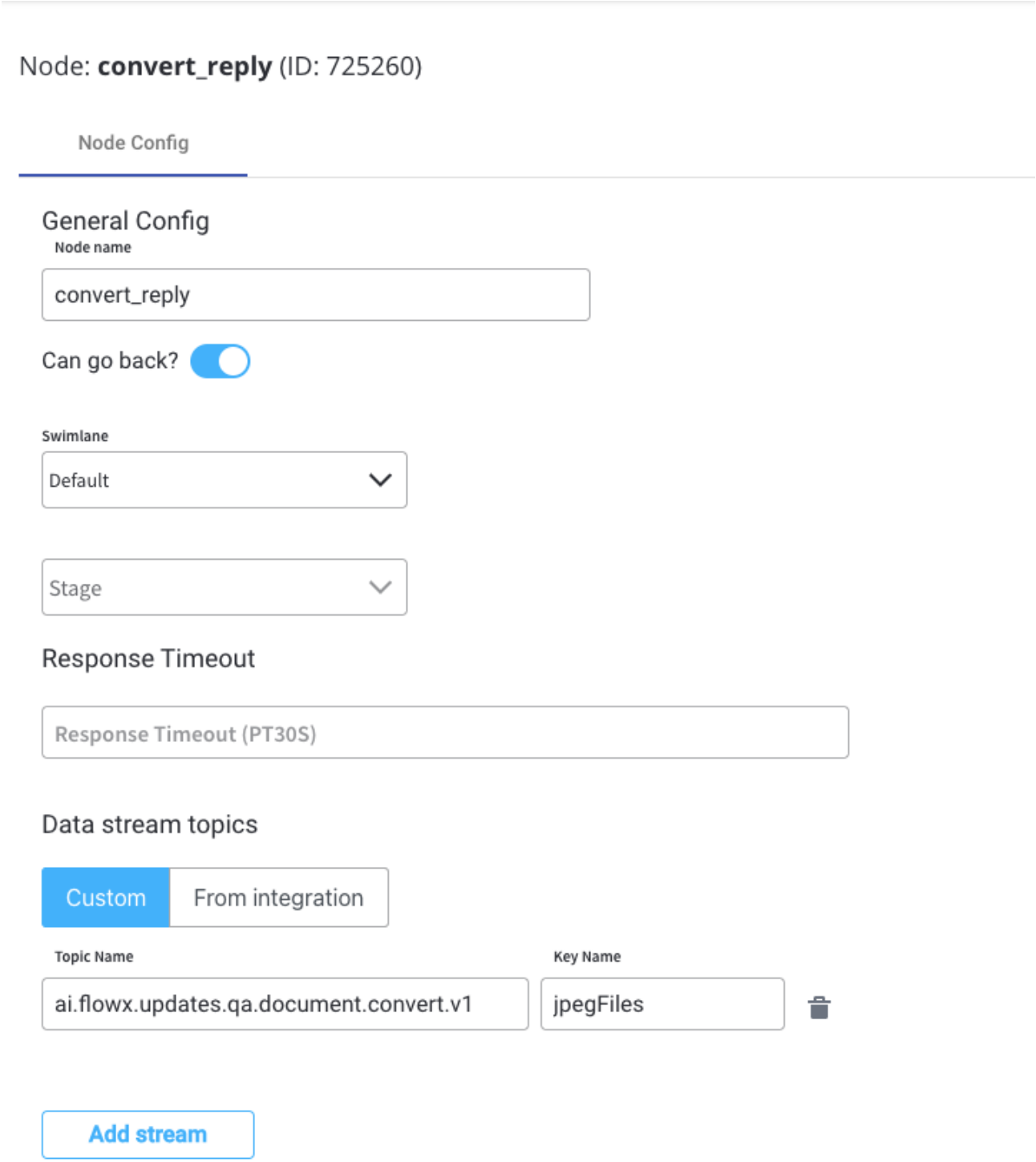
The fallback content to display on prerendering
, as it listens for messages on topics with specific names.

Receiving the reply

INFO

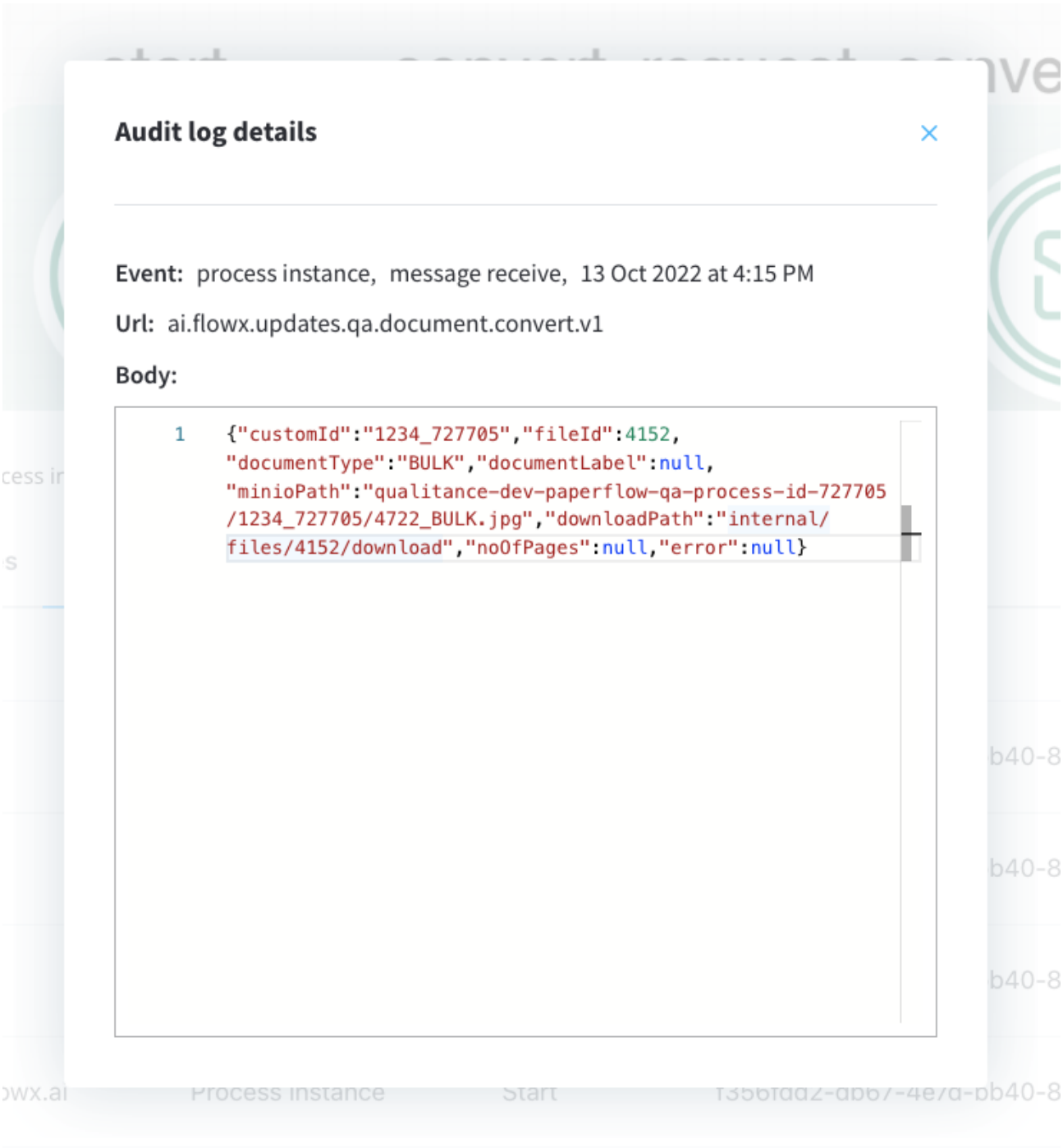
You can view the response by accessing the **Audit log** menu.

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?

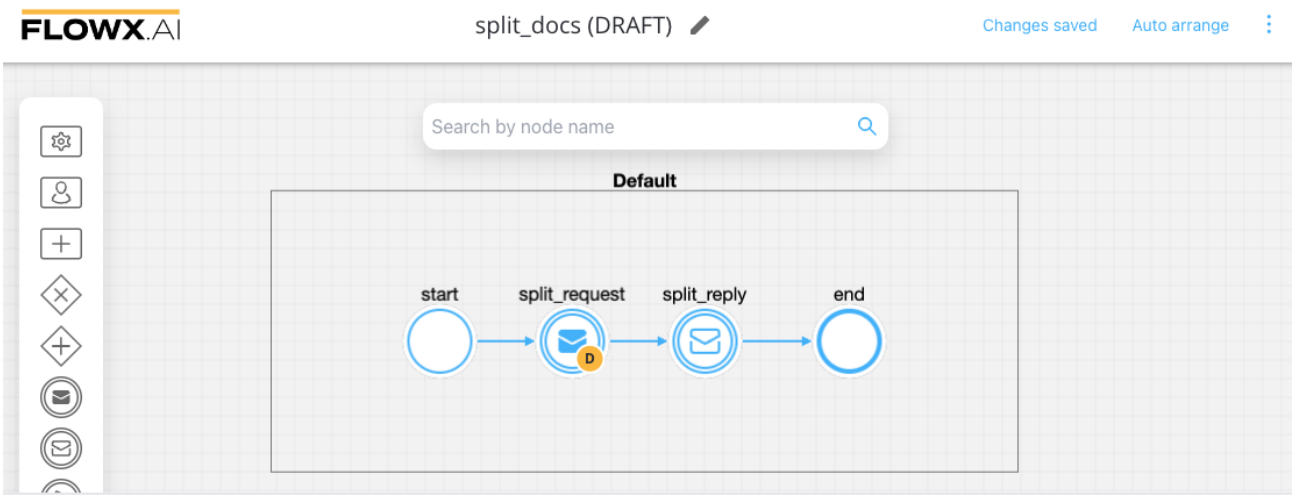
PLATFORM DEEP DIVE / Plugins / Custom Plugins / Documents plugin / Using the plugin / Splitting a document

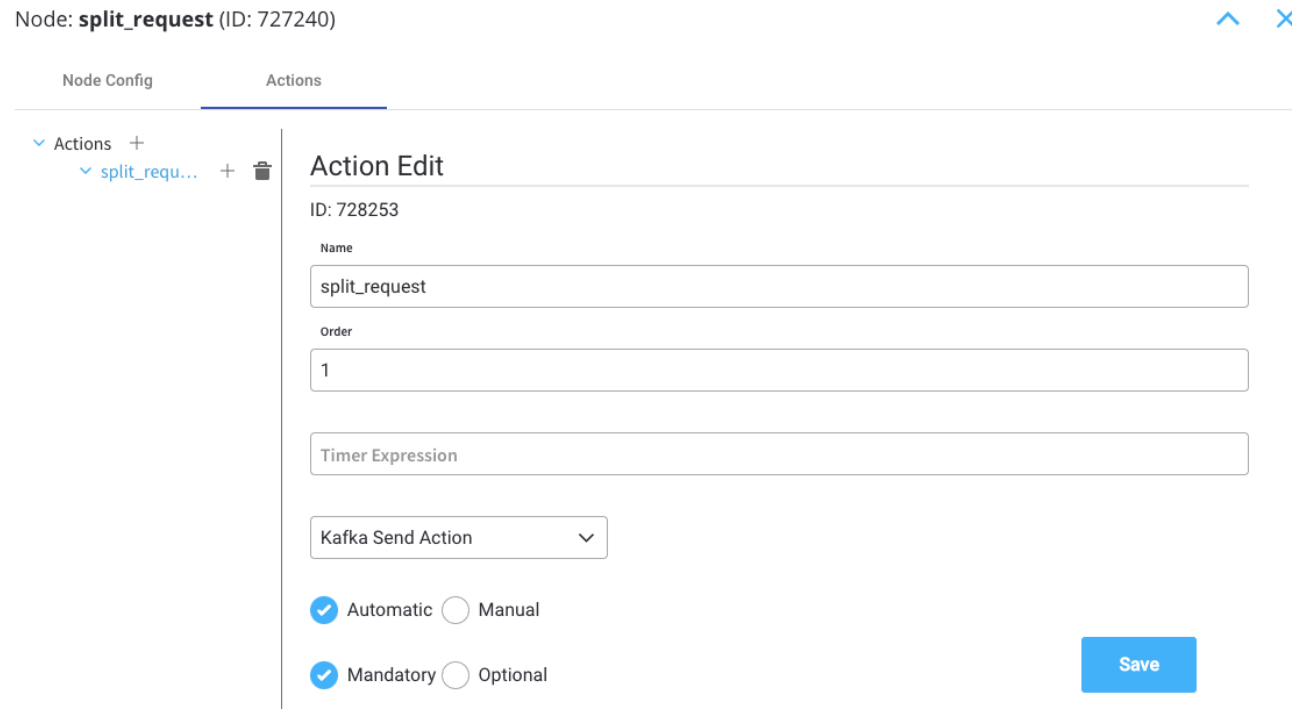
You can split a document into multiple parts using the Documents Plugin. This feature is useful, for example, when a user uploads a bulk scanned file that needs to be separated into separate files.

Sending the request

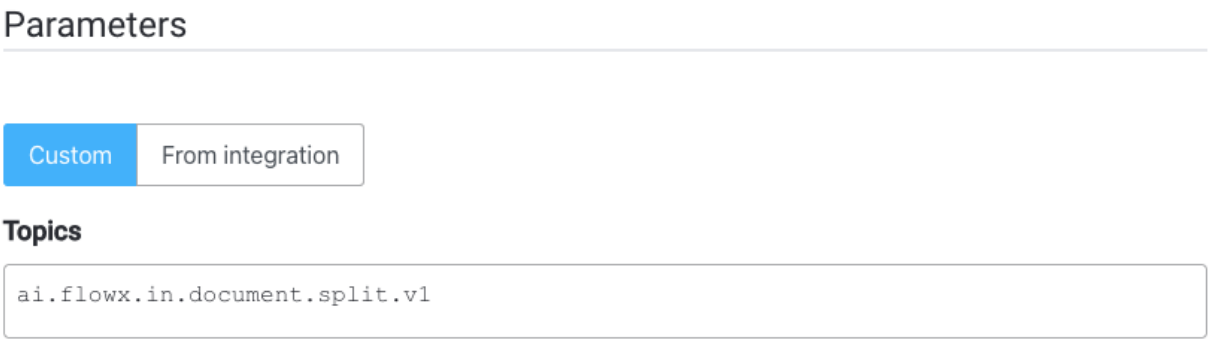
To split a document, follow these steps:

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.

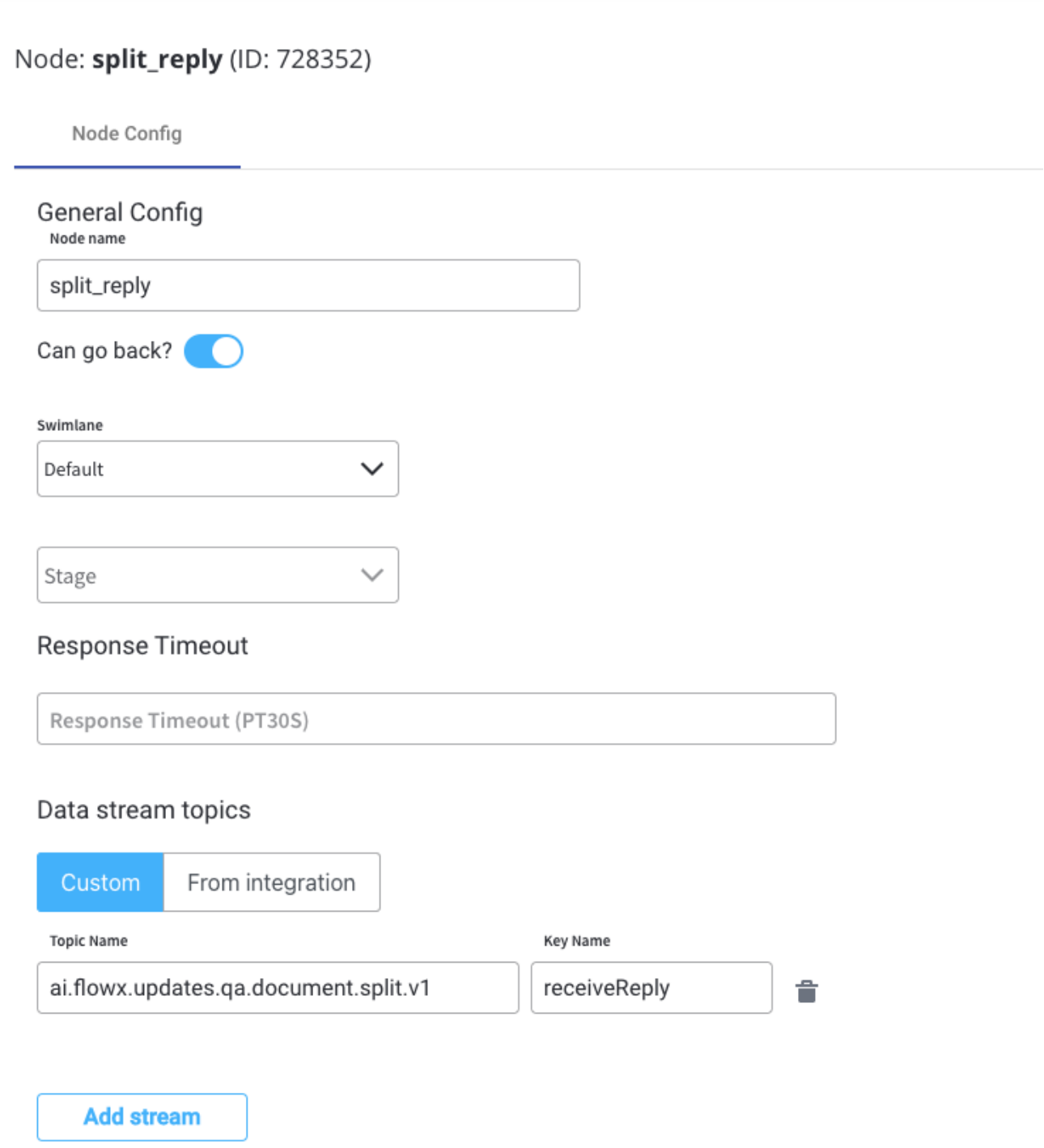
- **shouldOverride**: A boolean value (true or false) indicating whether to override an existing document if one with the same name already exists.
- **pagesNo**: The pages that you want to separate from the document.

> **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.

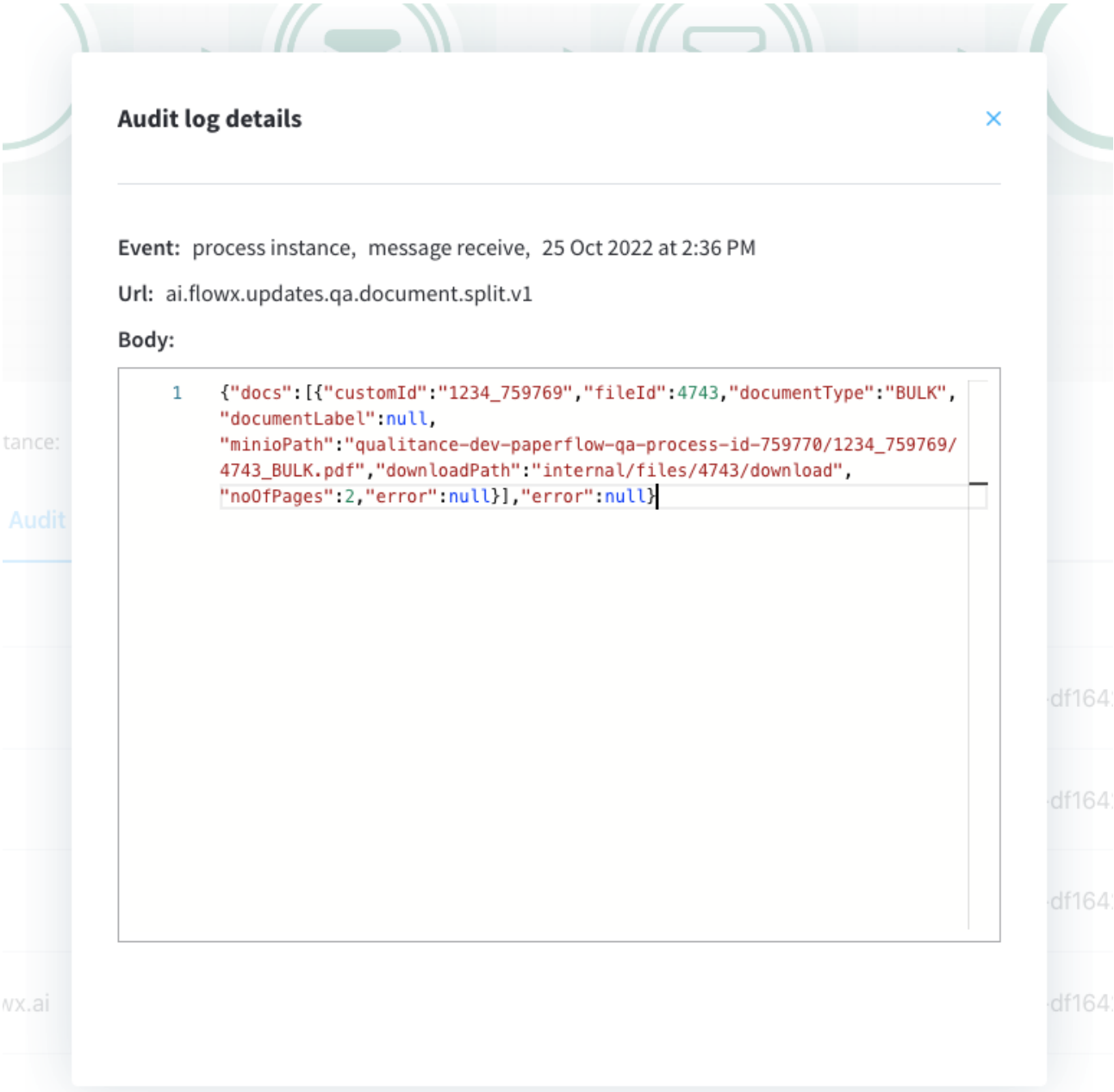
Receiving the reply

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

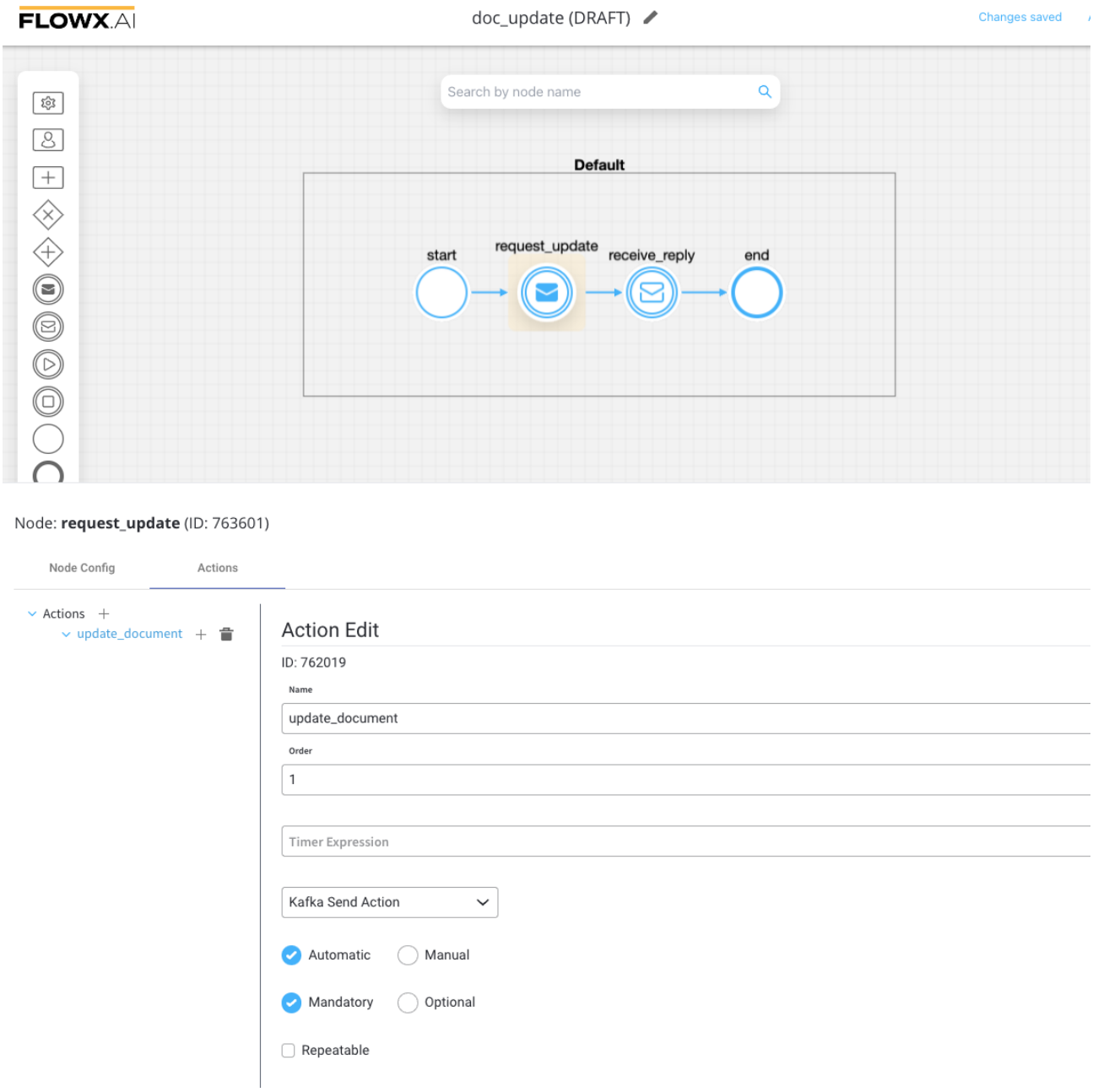
The documents plugin provides functionality for updating and deleting files associated with documents. You can update existing files or remove them from a document.

Updating files

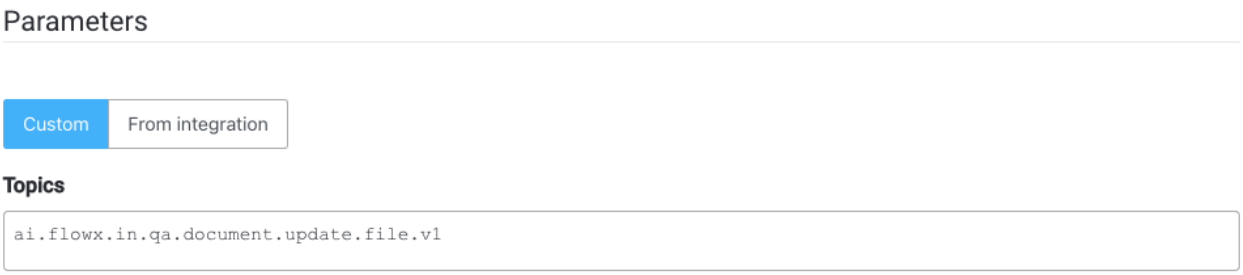
Sending the request

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:

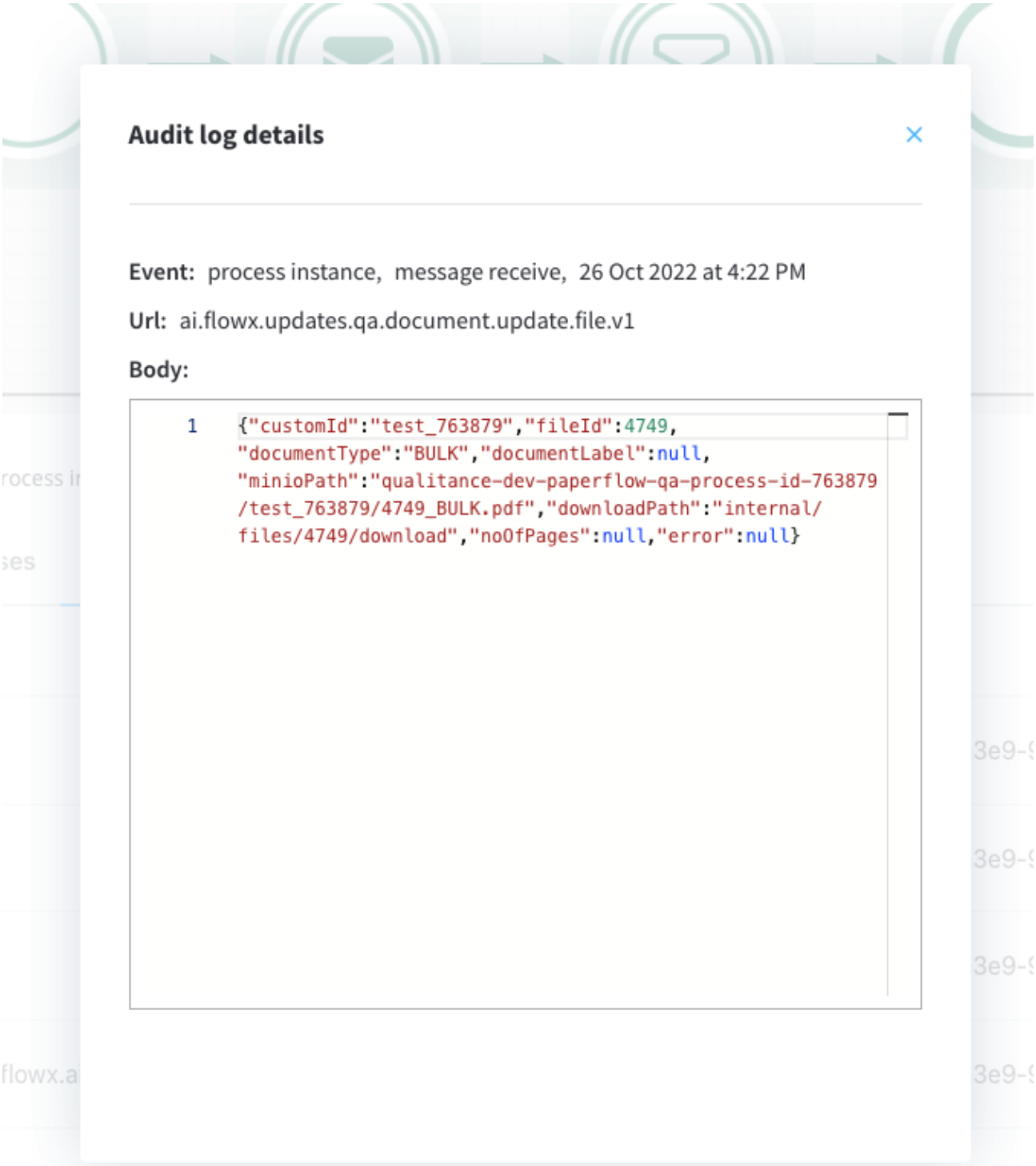
- `KAFKA_TOPIC_FILE_UPDATE_IN:` - default value: `ai.flowx.in.qa.document.update.file.v1`
- `KAFKA_TOPIC_FILE_UPDATE_OUT` - default value: `ai.flowx.updates.qa.document.update.file.v1`

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:

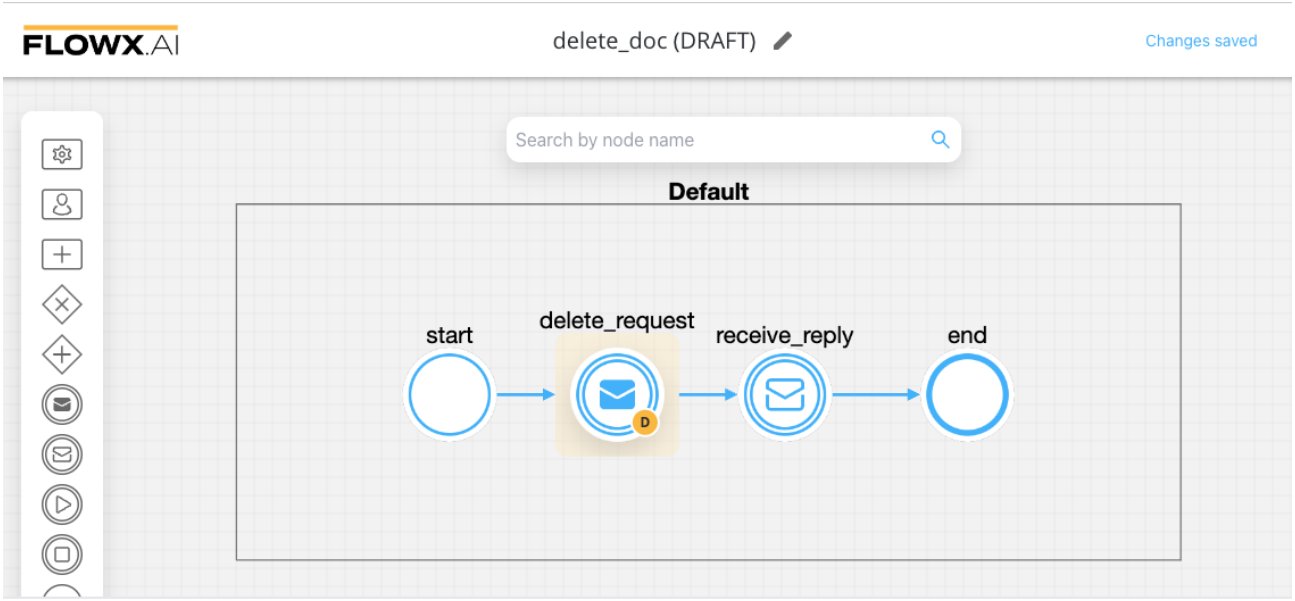

```
{
  "customId": "test_763879",
  "fileId": 4749,
  "documentType": "BULK",
  "documentLabel": null,
  "minioPath": "qualitance-dev-paperflow-qa-process-id-
763879/test_763879/4749_BULK.pdf",
  "downloadPath": "internal/files/4749/download",
  "noOfPages": null,
  "error": null
}
```

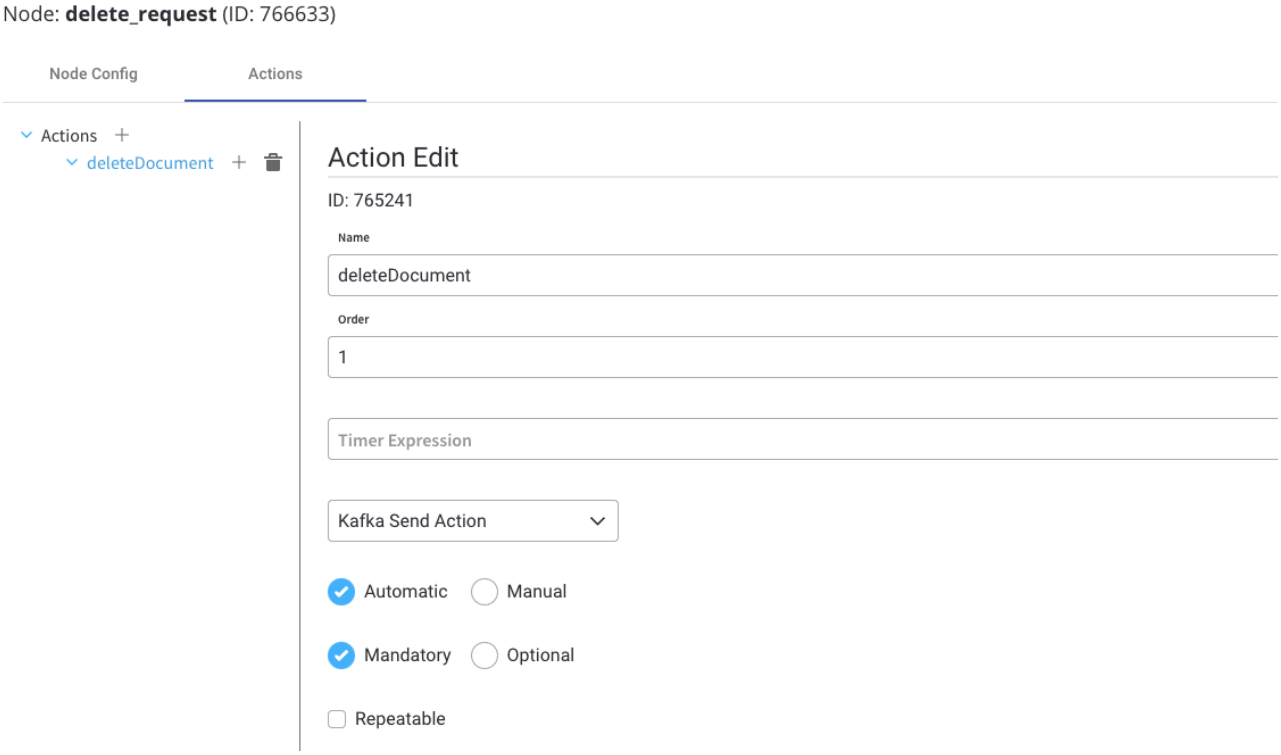

Deleting files from a document

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:

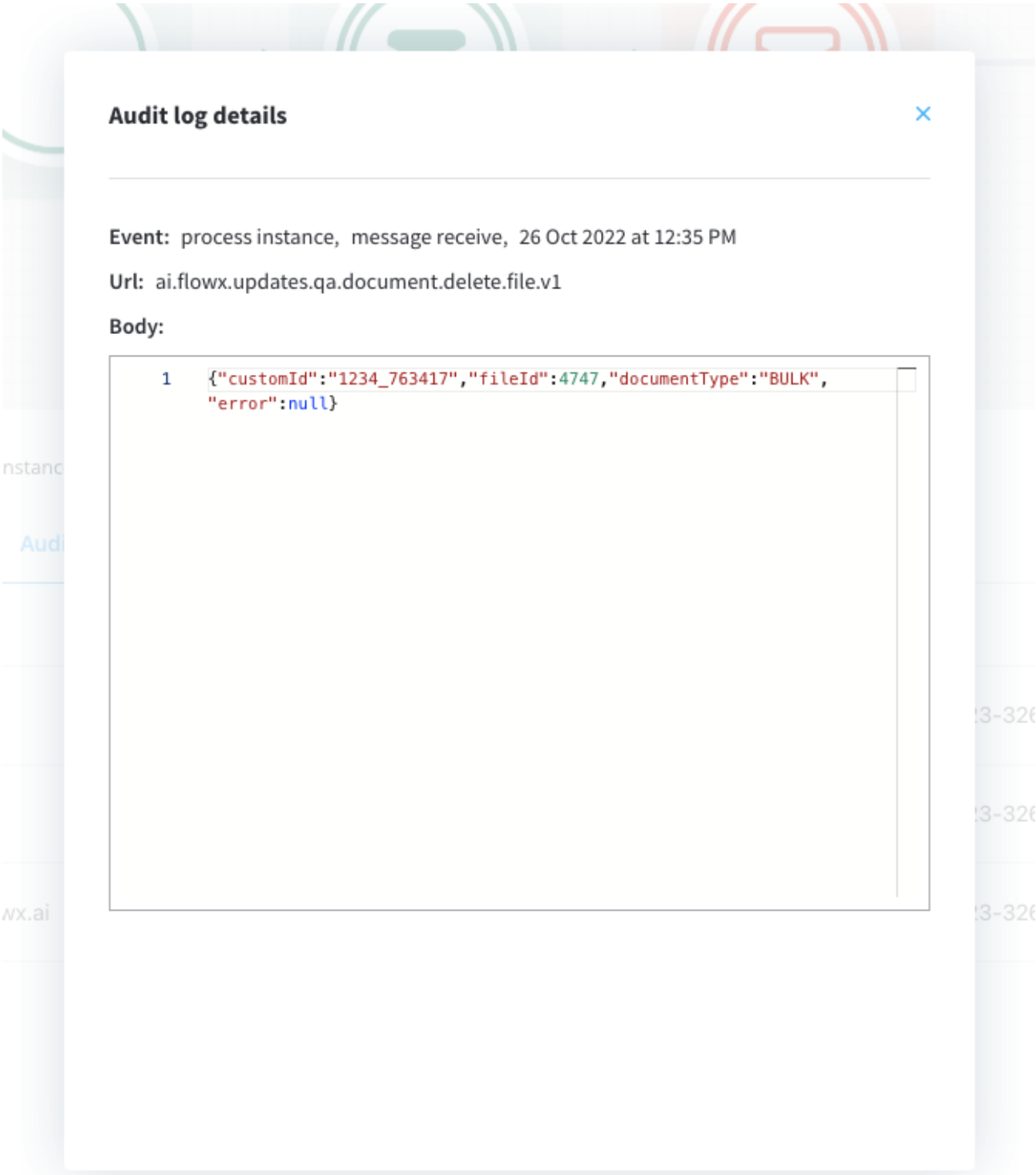
4. Fill in the body message request:

- `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

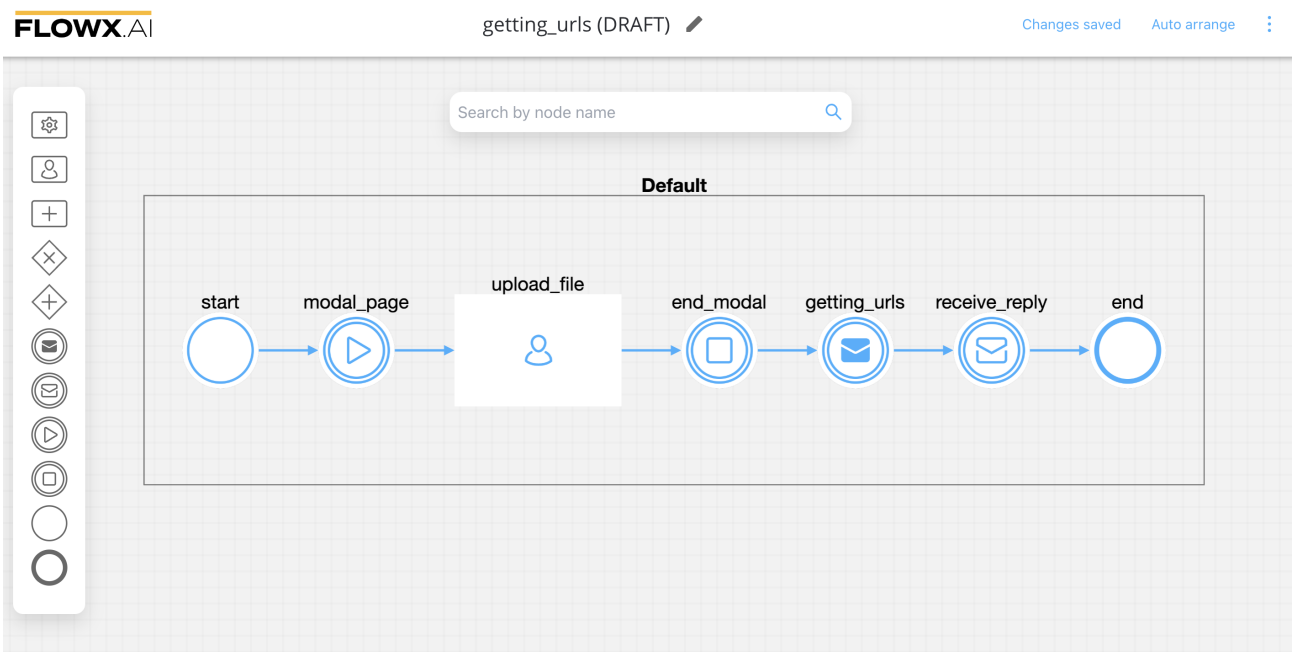
In certain scenarios, you may need to obtain URLs that point to uploaded documents to be used by other integrations. This requires adding a custom action to your process that requests the URLs from the Documents Plugin.

Sending the request

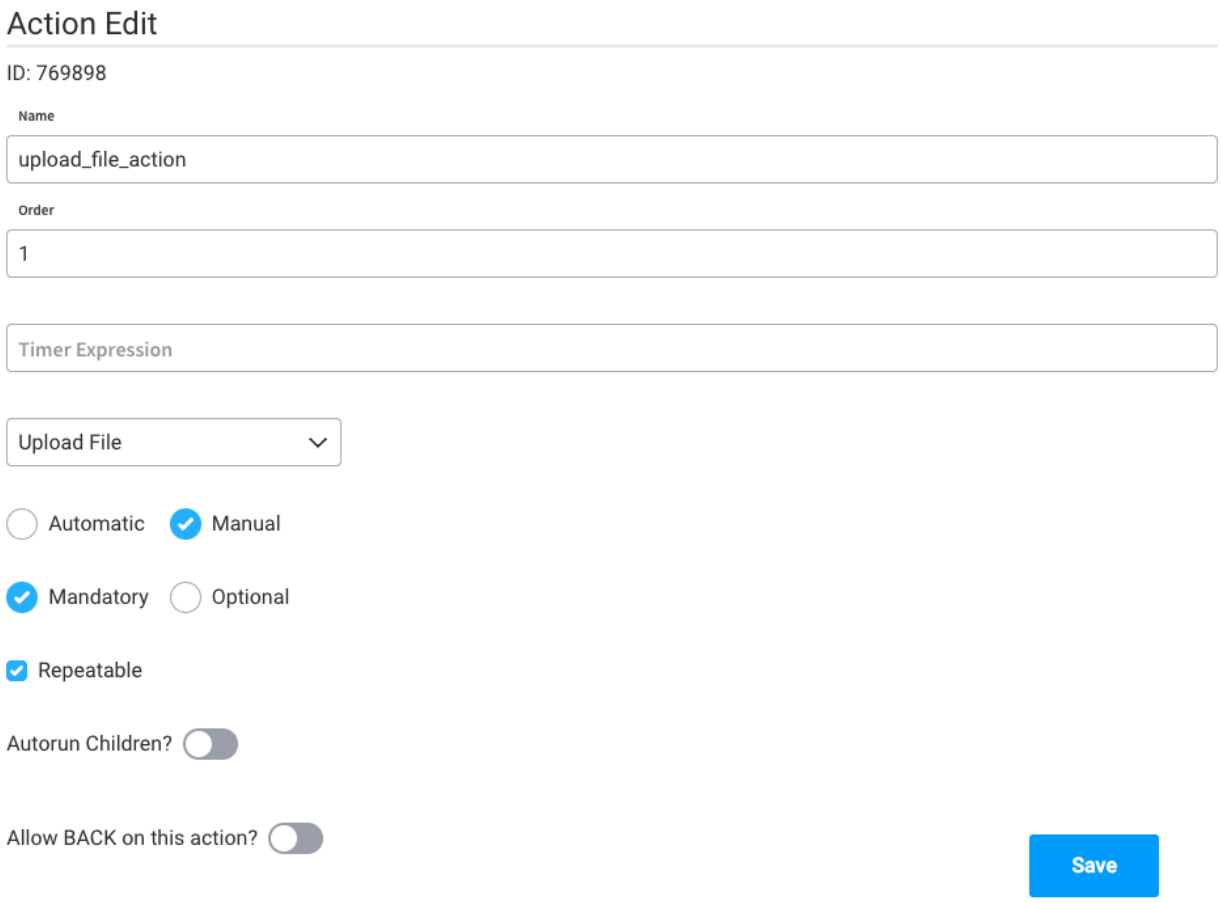
To retrieve document URLs and use them, for example, in the Notification Plugin to attach them to emails, follow the next steps:

1. Create a process and include the following nodes:

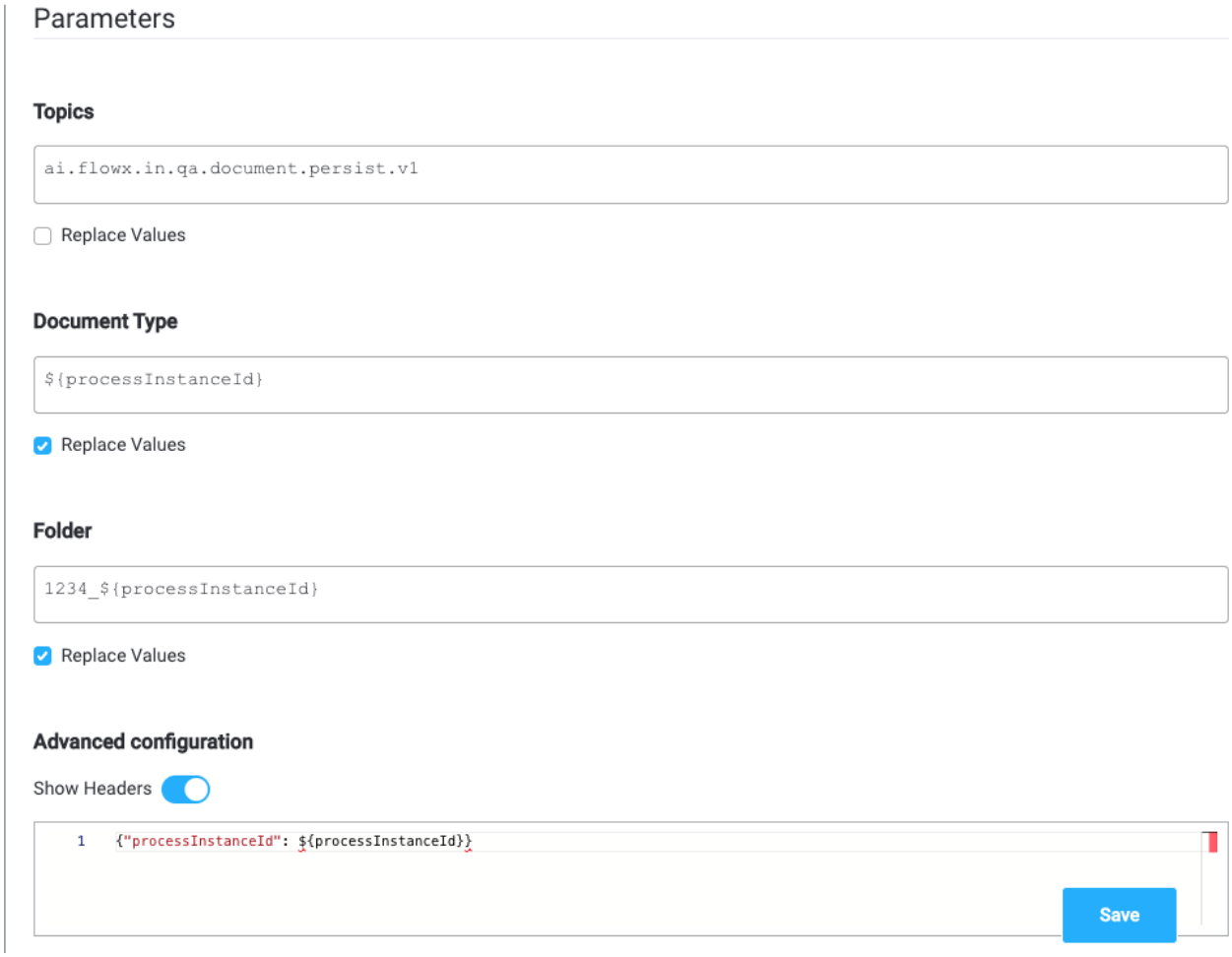
- a **Kafka Send Event Node**,
- a **Kafka Receive Event Node**
- a **User Task Node**
- **Start / End <ilestone** Nodes to create a modal



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

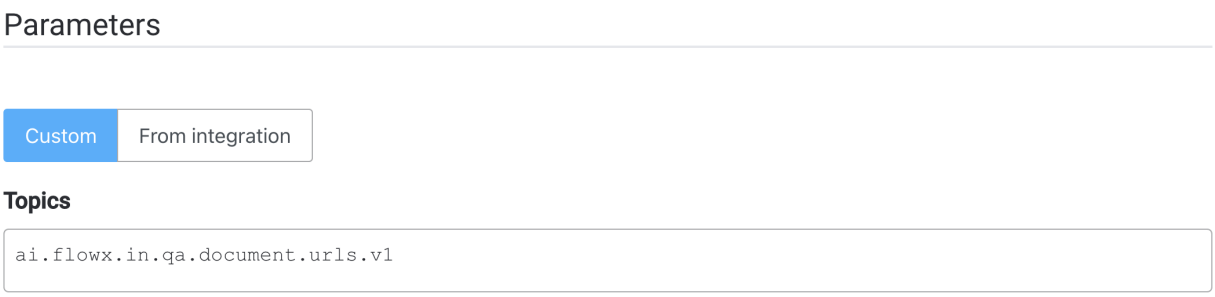


3. Configure the parameters for the **Upload Action**:

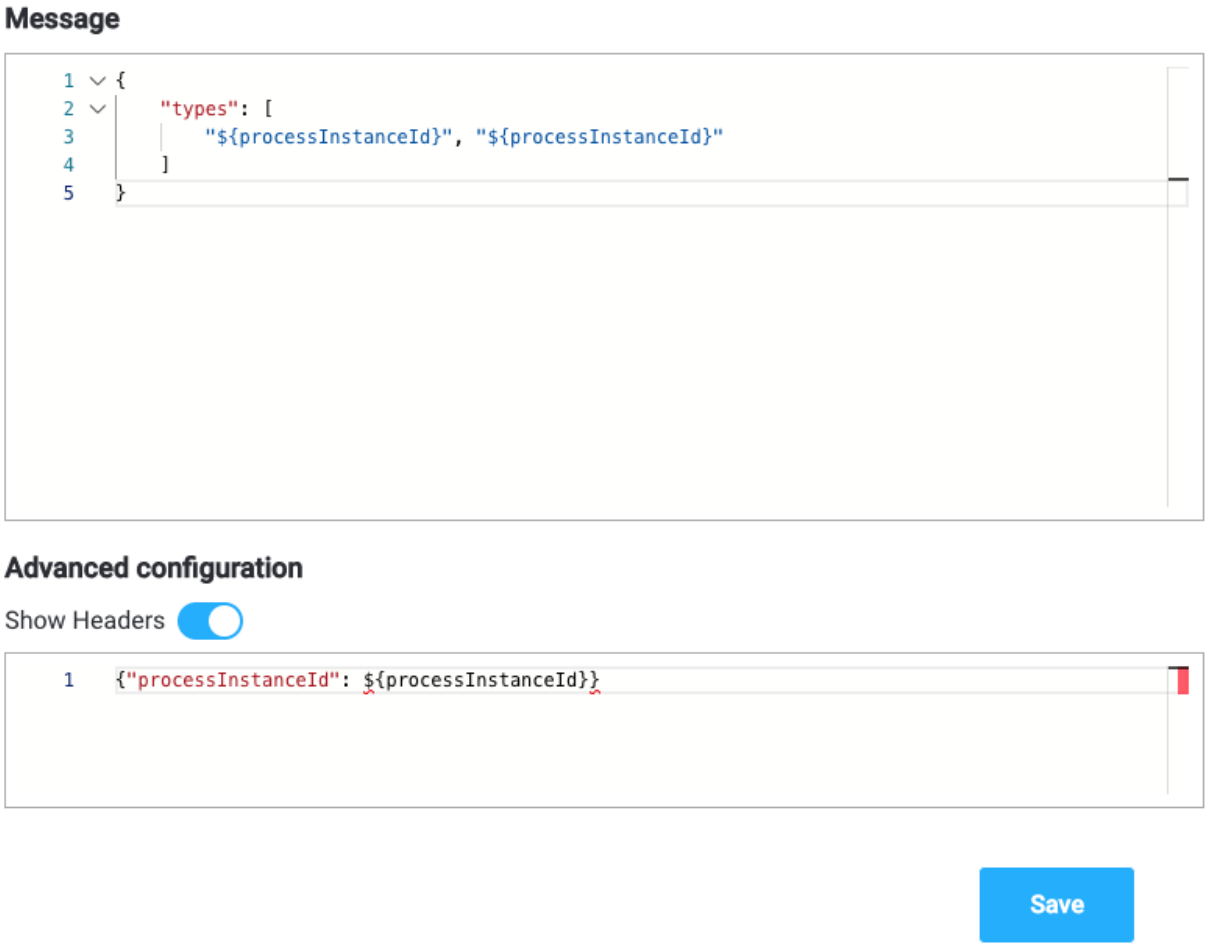


> **INFO**
>
> For more details on uploading a document and configuring an upload action, refer to the following sections:
>
> **Upload document**
>
> **Upload action**

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

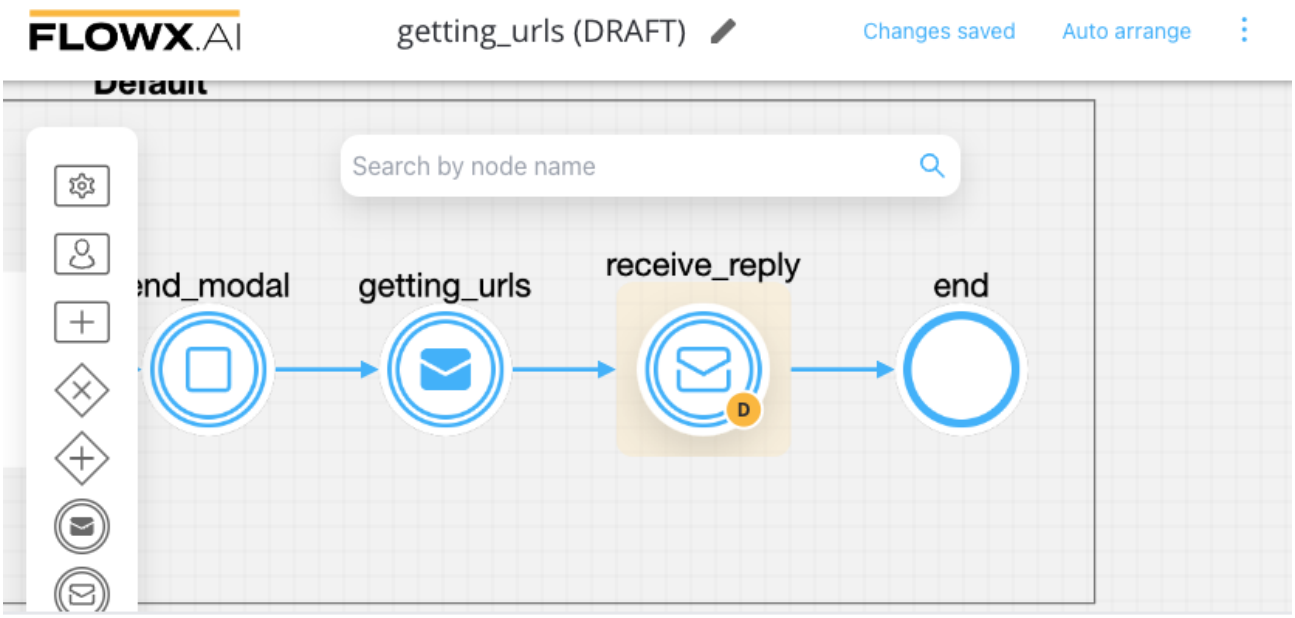


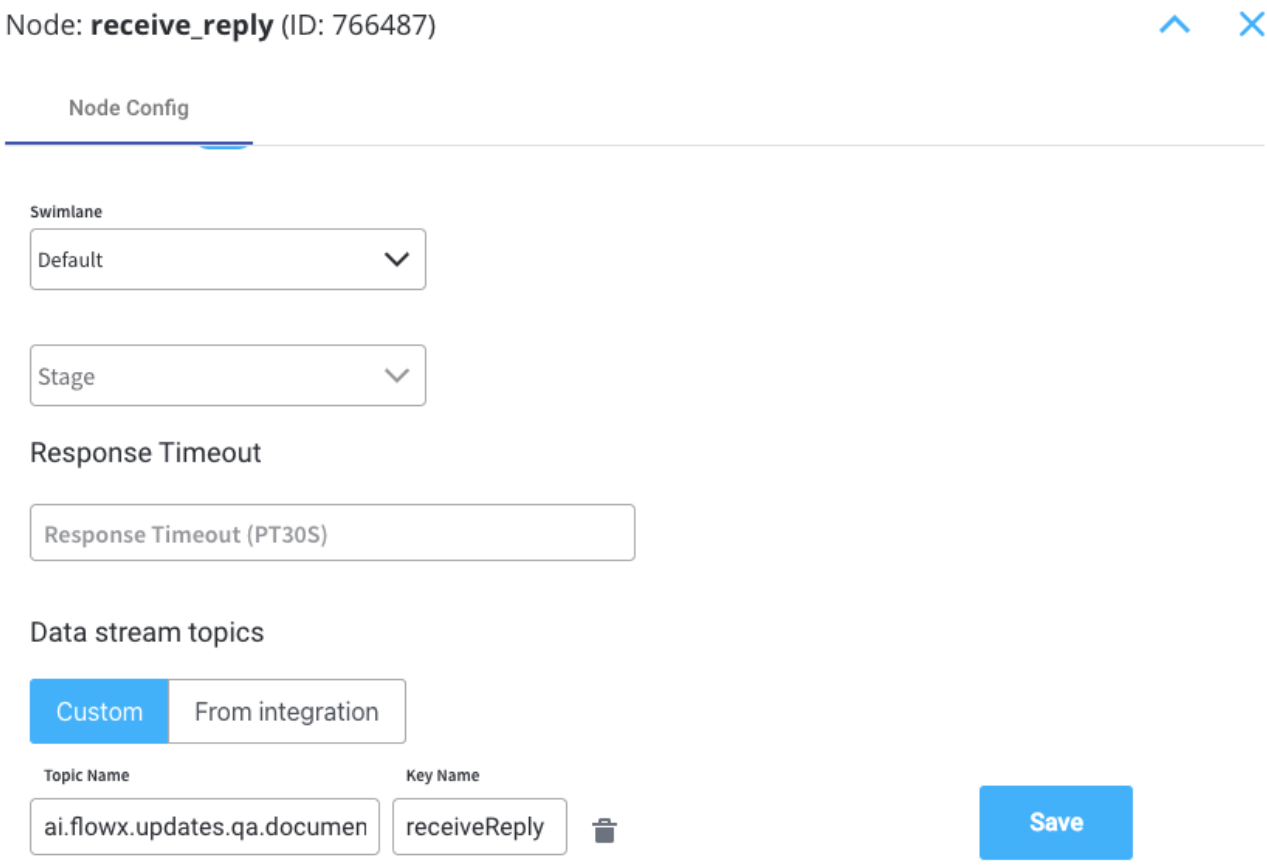
5. Fill in the body of the request message for the action:



- `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.


FLOWX.AI
getting_urls (DRAFT)
Changes saved
Auto arrange
Search by node name
end_modal
getting_urls
receive_reply
end
D


Node: receive_reply (ID: 766487)
Node Config
Swimlane
Default
Stage
Response Timeout
Response Timeout (PT30S)
Data stream topics
Custom
From integration
Topic Name
Key Name
ai.flowx.updates.qa.documen
receiveReply
Save

INFO

Kafka topic names can be set by using environment variables:

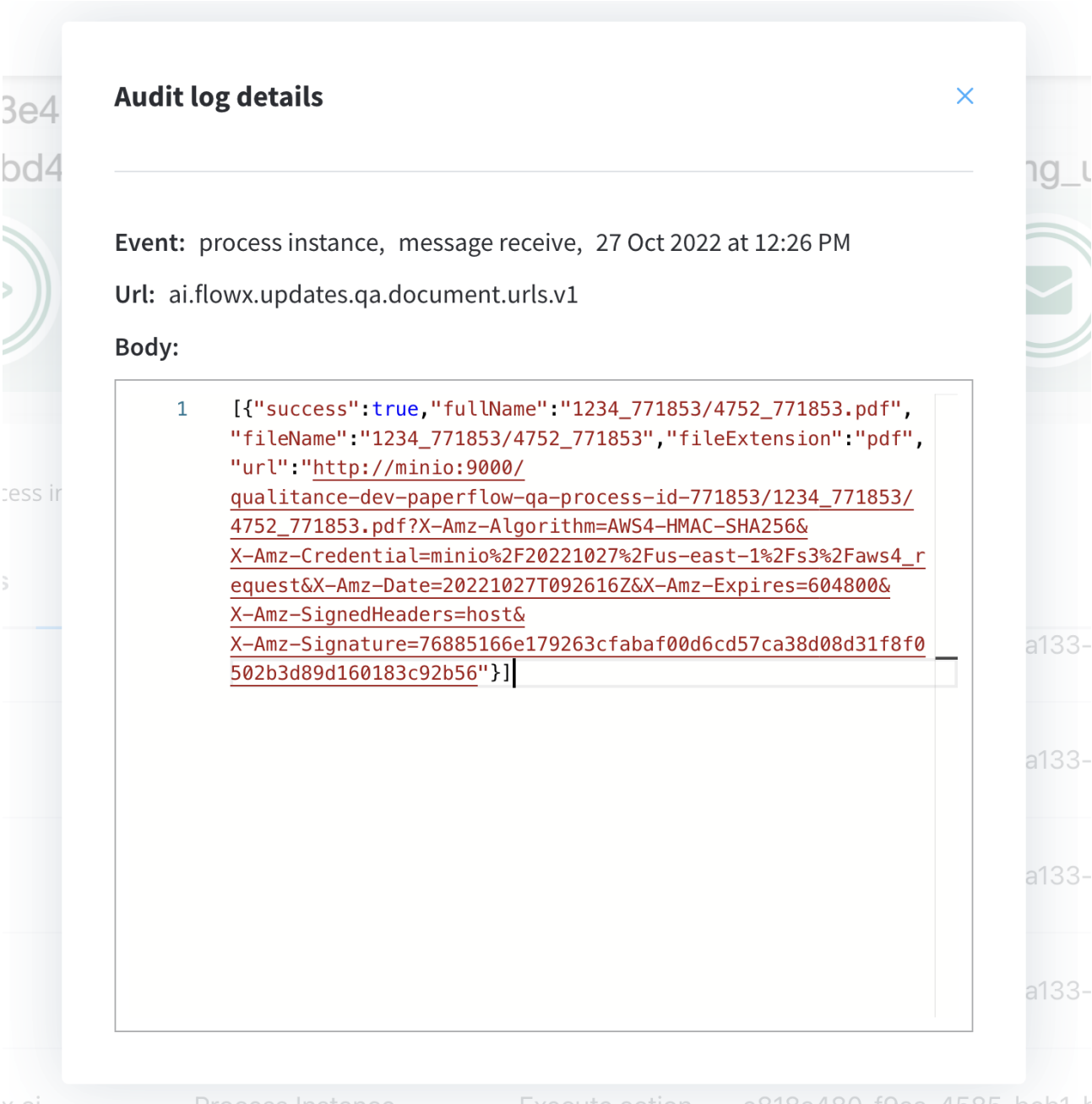
- `KAFKA_TOPIC_DOCUMENT_GET_URLS_IN` - `ai.flowx.in.qa.document.urls.v1` - the topic that listens for the request from the engine
- `KAFKA_TOPIC_DOCUMENT_GET_URLS_OUT` - `ai.flowx.updates.qa.document.urls.v1` - the topic on which the engine will expect the reply

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?