FLOWX.AI

PLATFORM DEEP DIVE / Core components / Core extensions / Content management

Contents

PLATFORM DEEP DIVE / Core components / Core extensions / Content management / Using the service

After you deployed the CMS service in your infrastructure, you can start defining and using custom content types, such as different lists (which can have different values for the same code depending on the external system that is used), blog posts etc.

You can also set the default application name to be used in your configuration. This is needed when retrieving the contents.

```
application:
    defaultApplication: DEFAULT_APPLICATION_NAME
```

If this configuration is not set, the service will use `flowx` as the default value.

Define needed Kafka topics

Kafka topic names can be set by using environment variables:

Default parameter (env var)	Default FLOWX.AI val
KAFKA_TOPIC_REQUEST_CONTENT_IN	ai.flowx.dev.plugin.cms.trigger.r

Default parameter (env var)	Default FLOWX.AI val
KAFKA_TOPIC_REQUEST_CONTENT_OUT	ai.flowx.dev.engine.receive.plug

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.

Example: Request a label by language or source system code

Used to translate custom codes into labels using the specified language or a certain source system.

Various external systems and integrations might use different labels for the same information. In the processes, it is easier to use the corresponding code and translate this into the needed label when necessary: for example when sending data to other integrations, when generating documents, etc.

You will need to add a Kafka send event CMS service.

The following values are expected in the request body:

- at least one of `language` and `sourceSystem` should be defined (if you only need the sourceSystem to be translated, you can leave language empty and vice versa, but they cannot both be empty)
- a list of `entries` to be translated

Example:

```
{
  "language": "en-US",
  "sourceSystem": "CS"
  "entries": [
    {
      "codes": [
        "ROMANIA",
        "BAHAMAS"
      ],
      "contentDescription": {
        "name": "country",
        "application": "flowx",
        "version": 1,
        "draft": true
      }
    }
  ]
}
```

If the value for `application` is not sent, the `defaultApplication` value will be used when retrieving the contents from the database.

`version` and `draft` are not mandatory, if they are not specified, the latest published content will be used.

The service will respond with the following message structure:

```
{
  "entries": [
    {
```

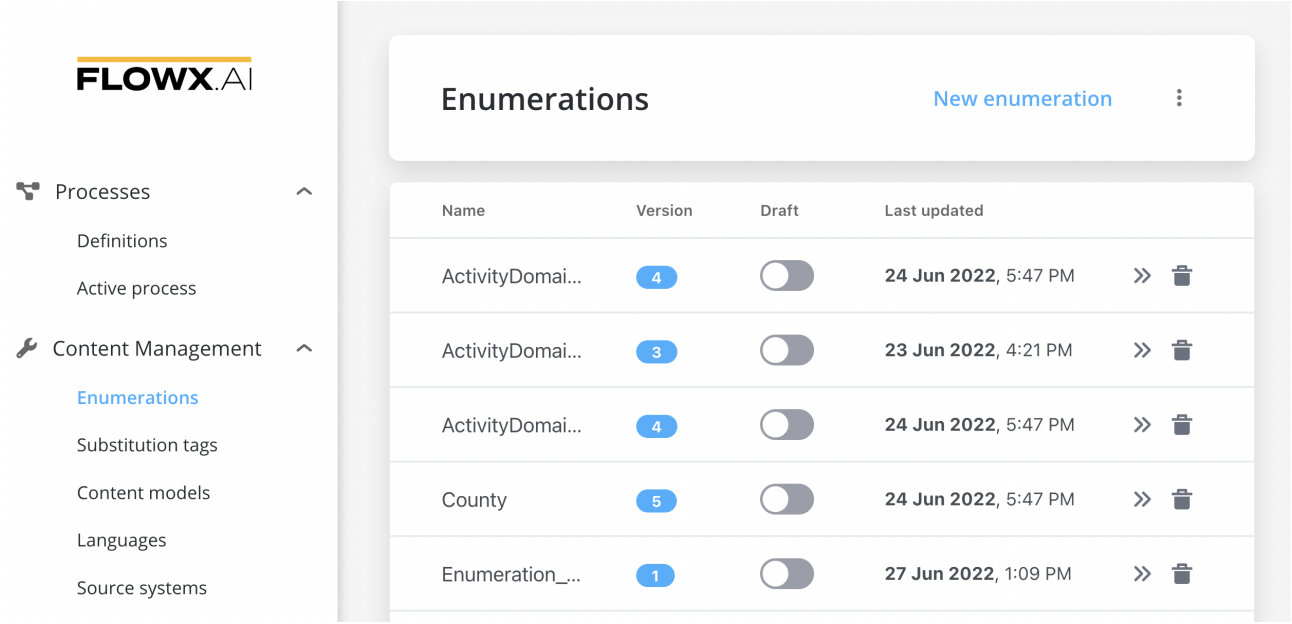
```
      "code": "ROMANIA",
      "label": "ROMANIA -en"
      "translatedCode": "ROMANIA-CS"
    },

    {
      "code": "BAHAMAS",
      "label": "BAHAMAS -en"
      "translatedCode": "BAHAMAS-CS"
    }
  ],
  "error": null
```


Was this page helpful?

PLATFORM DEEP DIVE / Core components / Core extensions / Content management / Enumerations

A collection of values that can be utilized as content in UI components or templates is managed using

The fallback content to display on prerendering
. Values can be defined for certain source systems or languages.

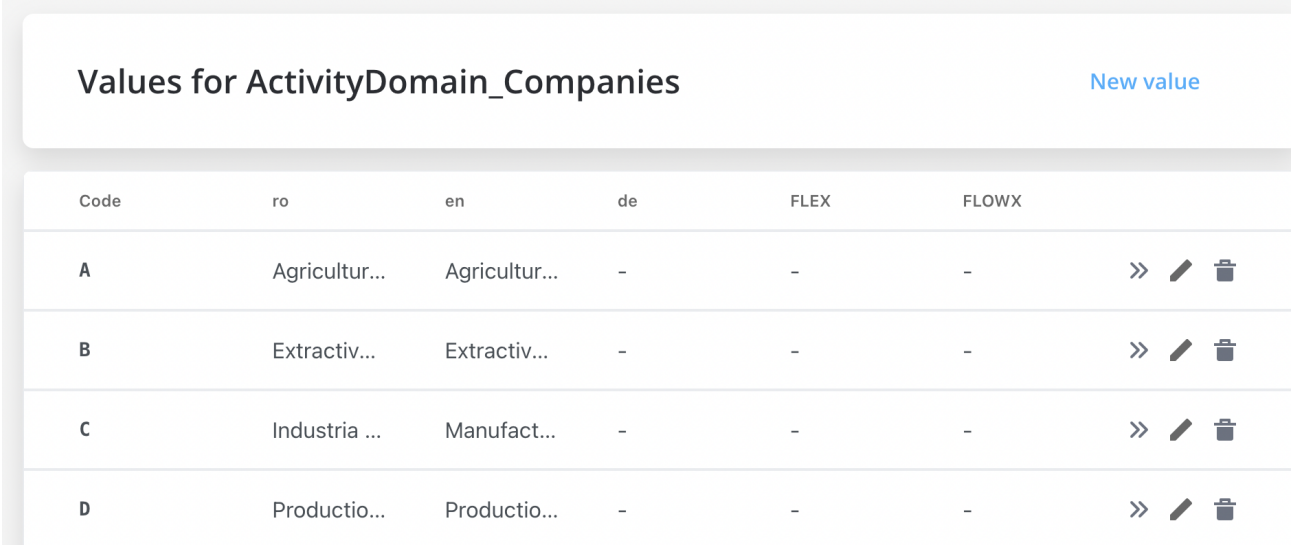


On the main screen inside **Enumerations**, you have the following elements:

- **Name** - the name of the enumeration
- **Version** - the version of the enumeration
- **Draft** - switch button used to control the status of an enumeration, could be **Draft** or **Published**
- **Last Updated** - the last time an enumeration has been updated
- **Open** - button used to access an enumeration to configure it/ add more values, etc.
- **Delete** - button used to delete an enumeration
- **New enumeration** - button used to create a new enumeration
- **Breadcrumbs** >** **Import/Export**

For each entry (when you hit the **Open** button) inside an enumeration we have to define the following properties:

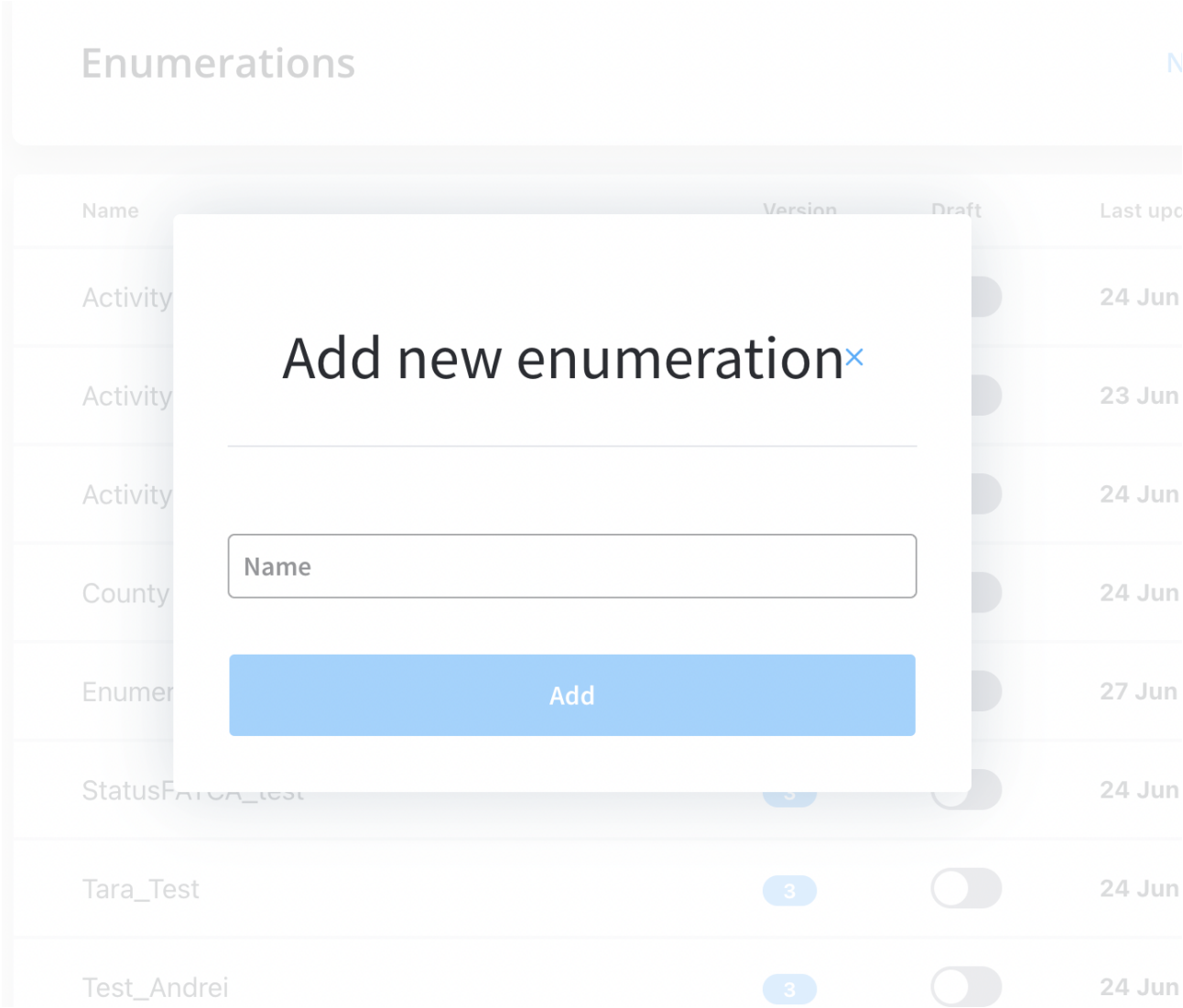
- **Code** - not displayed in the end-user interface, but used to assure value uniqueness
- **Labels** - strings that are displayed in the end-user interface, according to the language set for the generated solution
- **External source systems codes** - values that are set for each external system that might consume data from the process; these codes are further used by connectors, in other to send to an external system a value that it can validate



Adding a new enumeration

To add a new enumeration, follow the next steps:

1. Go to **FLOWX Designer** and select the **Content Management** tab.
2. Select **Enumerations** from the list.
3. Add a suggestive name for your enumeration and then click **Add**.



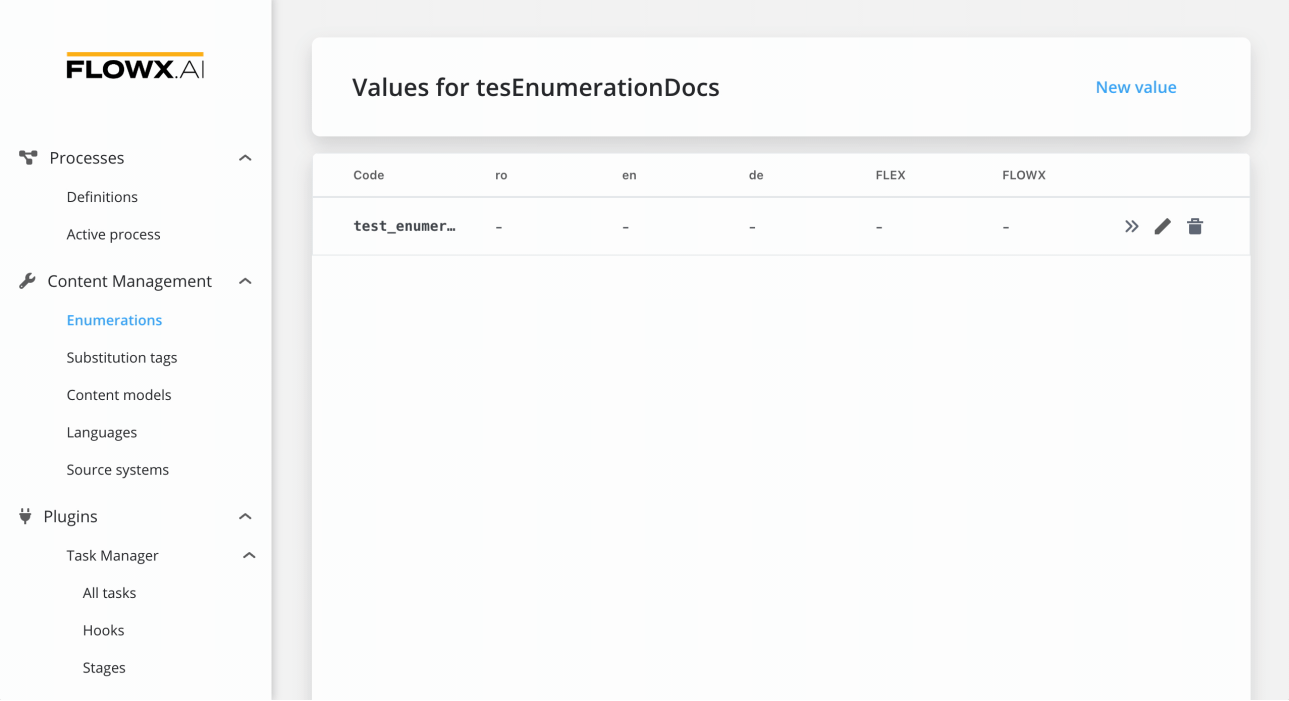
Configuring an enumeration

After creating an enumeration, you can add values to it.

To configure an enumeration value, follow the next steps:

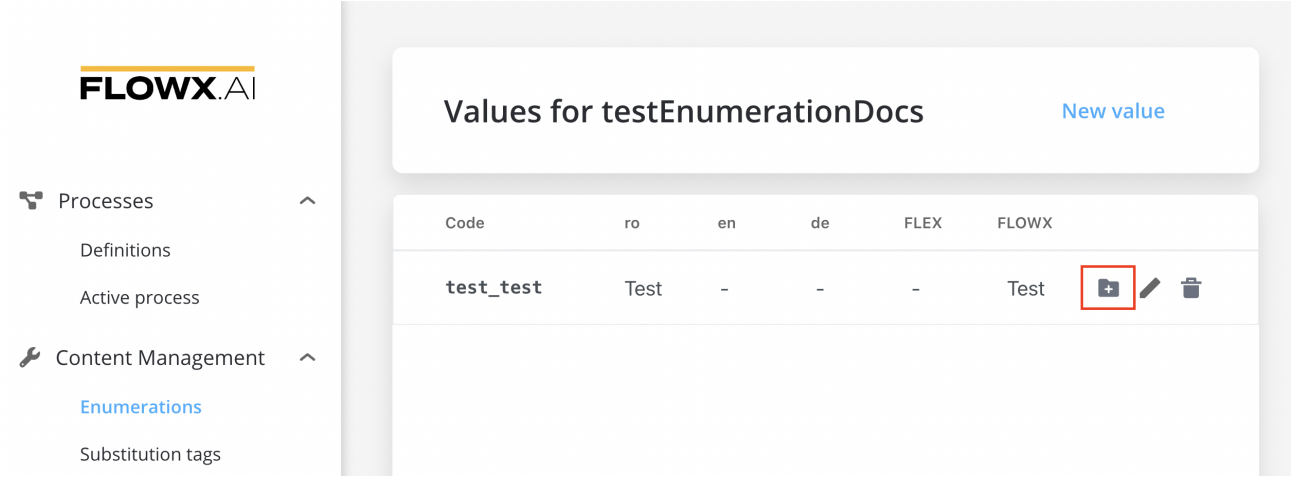
1. Go to FLOWX.AI Designer and select the **Content Management** tab.
2. Select **Enumerations** from the list and open an enumeration.
3. Click **New value** and fill in the necessary details:

- **Code** - as mentioned above, this is not displayed in the end-user interface but is used to assure value uniqueness
- **Labels** - set the value of the string for each language you would like to use
- **Source Systems** - values that are set for each external system that might consume data from the process



Creating a child collection

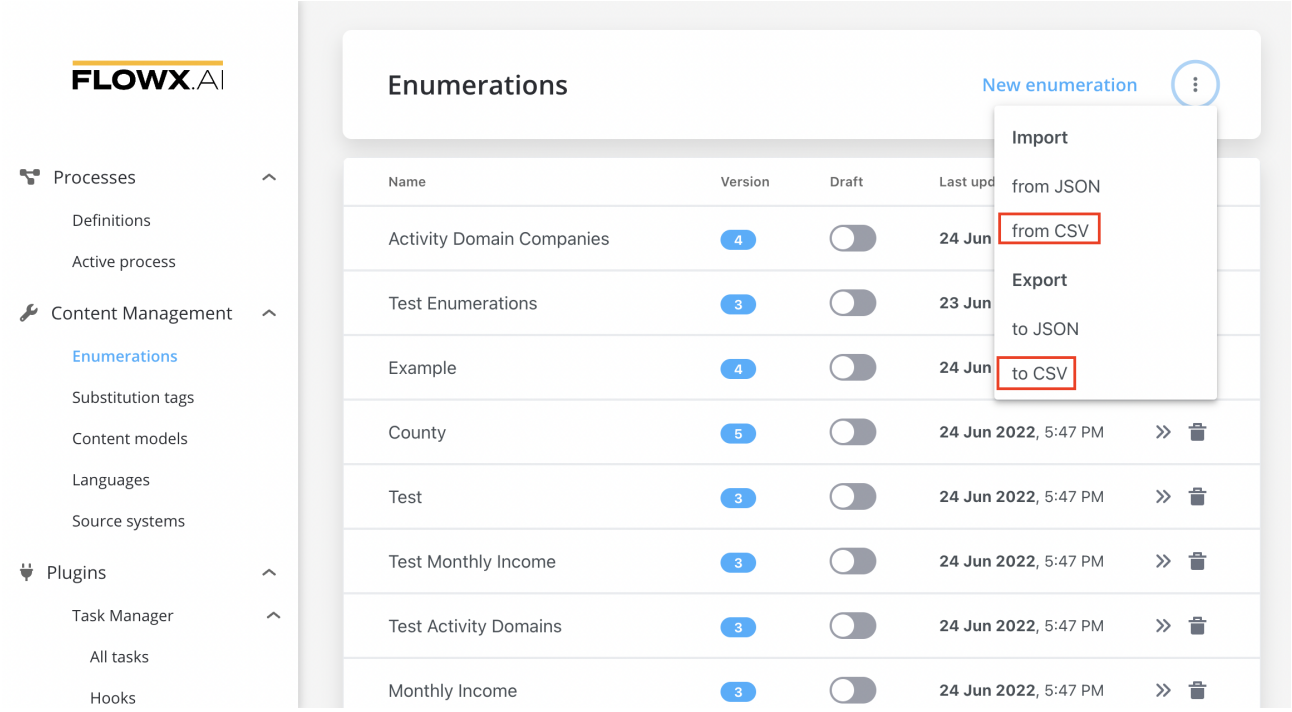
Enumerations can also be defined as a hierarchy - for each entry, we can define a list of children values (for example, name of the countries defined under the continents' enumeration values); hierarchies further enable cascading values in the end-user interface (for example, after selecting a continent in the first select UI component, the second select component will contain only the children of this continent).



Importing/exporting an enumeration

You can use the import/export feature to import or export enumerations using the following formats:

- JSON
- CSV



Enumerations example

Enumerations, for instance, can be used to build elaborate lists of values (with children). Assuming you wish to add enumerations for Activity Domain Companies, you can create children collections by grouping lists and other related domains and activities.

We have the following example for ***Activity Domain Companies***:

Activity Domain Companies → Agriculture forestry and fishing:

- **Agriculture, hunting, and related services →**

> Cultivation of non-perennial plants:

▸ Cultivation of plants from permanent crops:

▸ Animal husbandry:

- **Forestry and logging →**

▸ Forestry and other forestry activities:

▸ Logging:

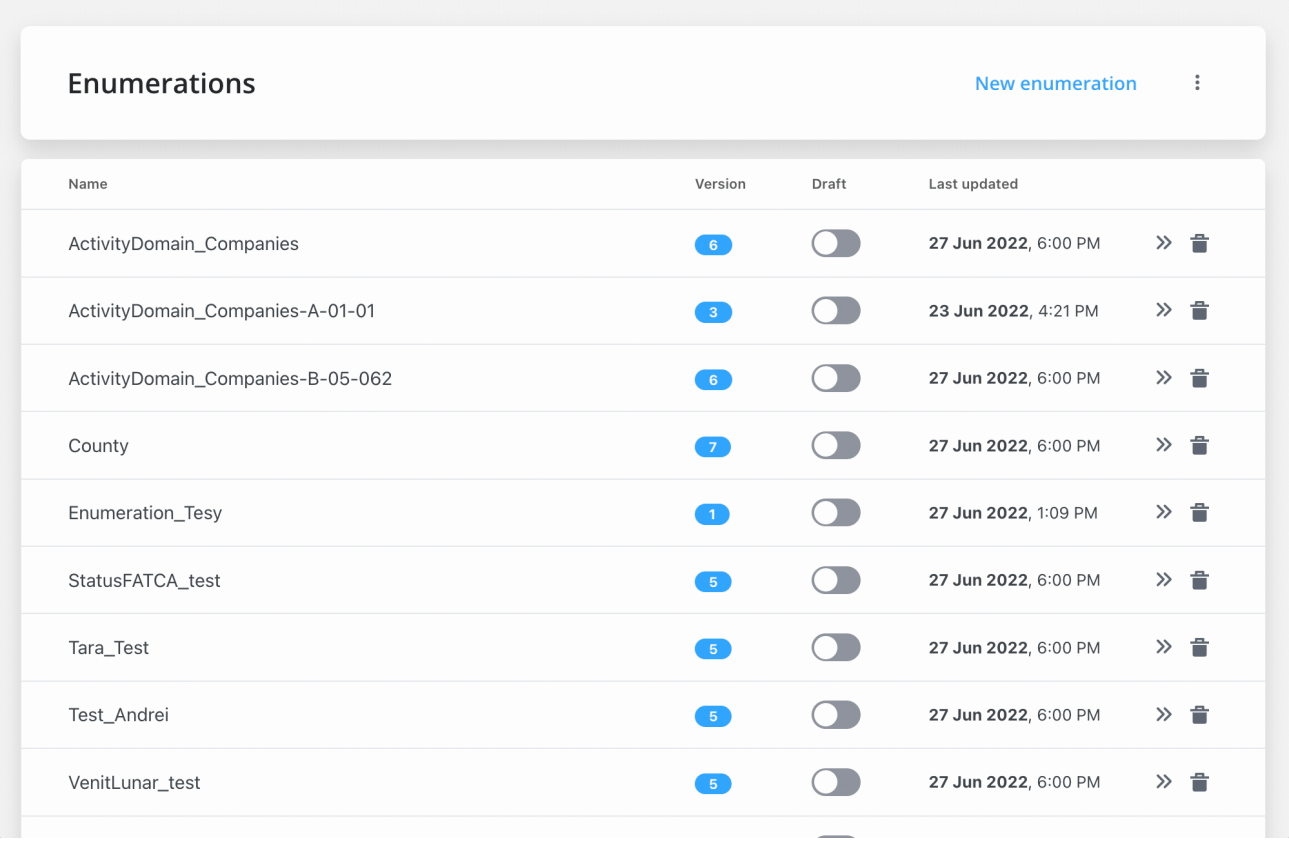
▸ Collection of non-wood forest products from spontaneous flora:

- **Fisheries and aquaculture →**

▸ Fishing:

▸ Aquaculture:

This is the output after adding all the lists/collections from above:



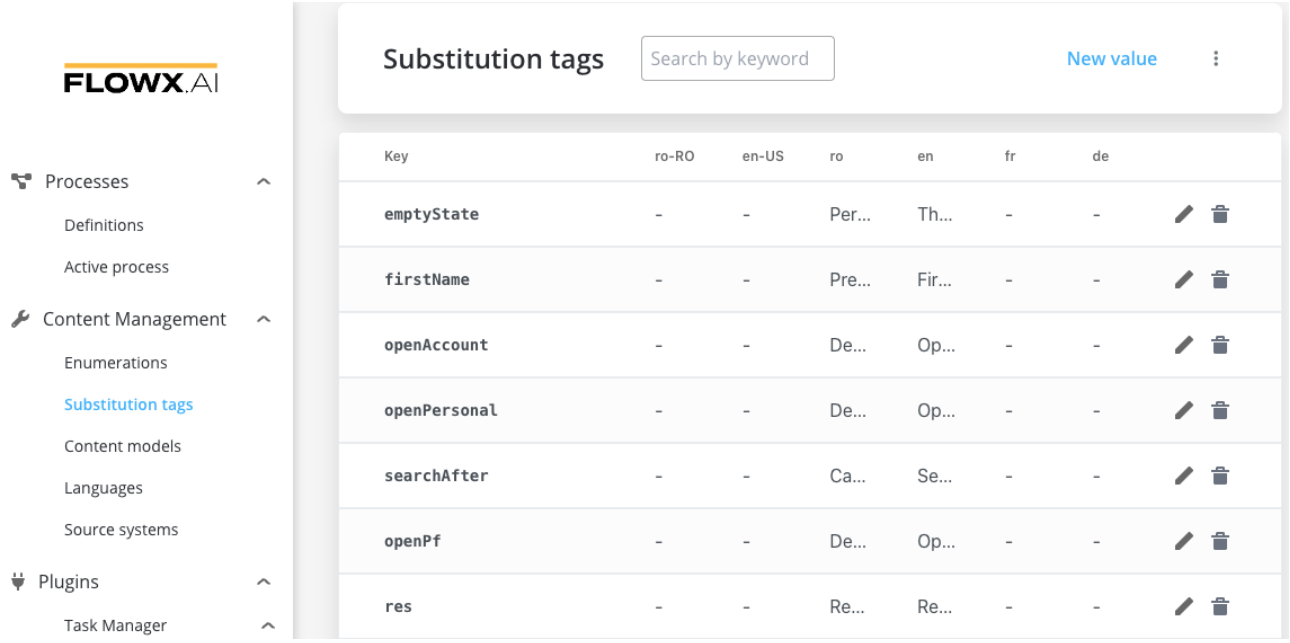
Was this page helpful?

PLATFORM DEEP DIVE / Core components / Core extensions / Content management / Substitution tags

Substitution tags are used to generate dynamic content across the platform. As

The fallback content to display on prerendering

, substitution tags can be defined for each language set for the solution.



On the main screen inside **Substitution tags**, you have the following elements:

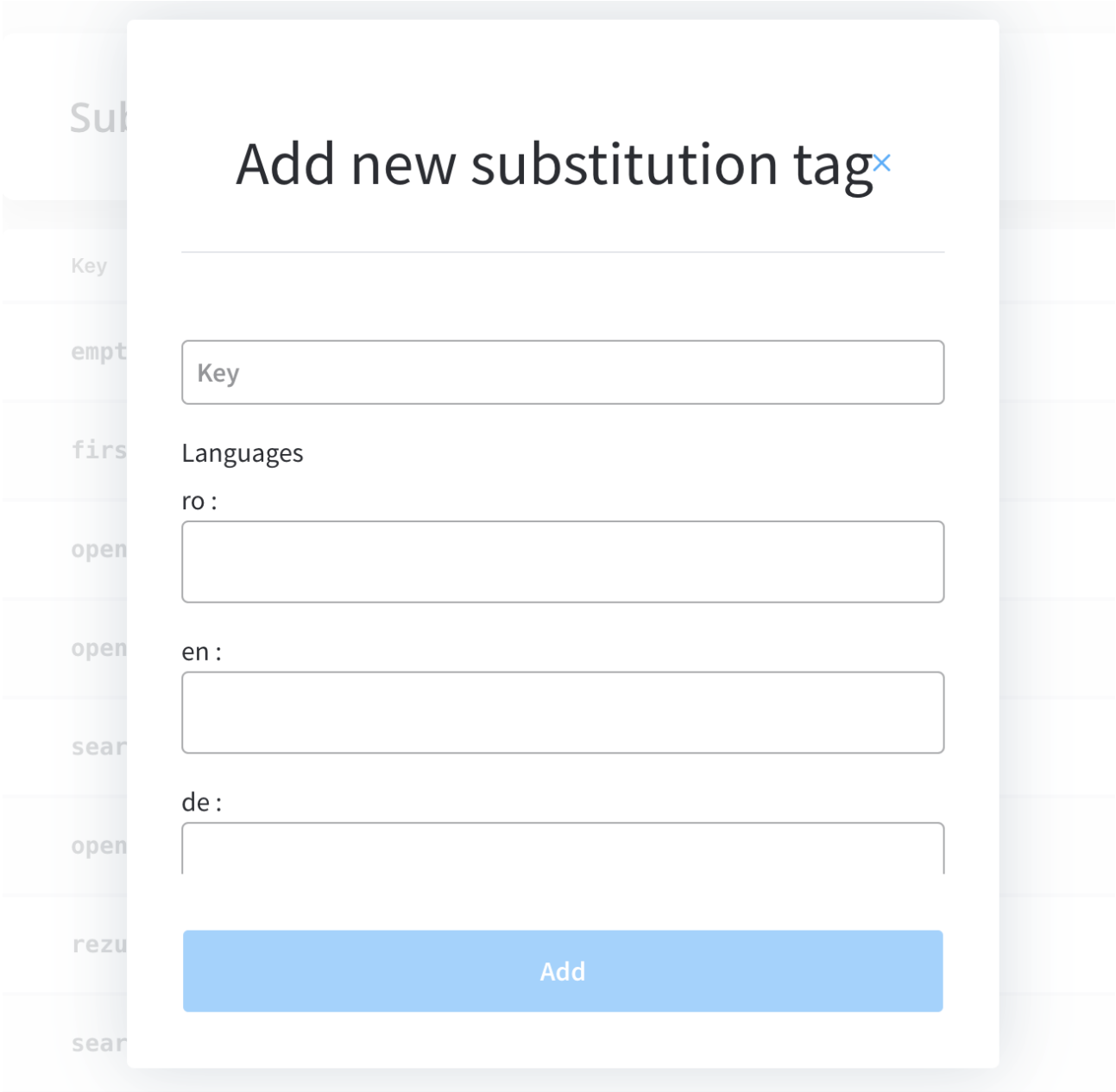
- **Key**
- **Values** - strings that are used in the end-user interface, according to the language set for the generated solution
- **Edit** - button used to edit substitution tags
- **Delete** - button used to delete substitution tags
- **New value** - button used to add a new substitution tag
- **Breadcrumbs menu**:
  - **Import**
    - from JSON
    - from CSV
  - **Export**
    - to JSON

    - to CSV
- **Search by** - search function used to easily look for a particular substitution tag

Adding new substitution tags

To add a new substitution tag, follow the next steps.

1. Go to
   The fallback content to display on prerendering
   and select the **Content Management** tab.
2. Select **Substitution tags** from the list.
3. Click **New value**.
4. Fill in the necessary details:
   - Key
   - Languages
5. Click **Add** after you finish.



CAUTION

When working with substitution tags or other elements that imply values from other languages defined in the CMS, when running a

The fallback content to display on prerendering

, the default values extracted will be the ones marked by the default language.

Getting a substitution tag by key

```
public func getTag(withKey key: String) -> String?
```

All substitution tags will be retrieved by the SDK before starting the first process and will be stored in memory.

Whenever the container app needs a substitution tag value for populating the UI of the custom components, it can request the substitution tag using the method above, providing the key.

For example, substitution tags can be used to localize the content inside an application.

Example

Localizing the app

> **INFO**
>
> You must first check and configure the FLOWX.AI Angular renderer to be able to replicate this example. Click here for more information.

The `flxLocalize` pipe is found in the `FlxLocalizationModule`.

```
import { FlxLocalizationModule } from 'flowx-process-
renderer';
```

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-dummy-component',
  template: ` <h3>{{ "stringToLocalize" | flxLocalize}}
</h3>`,

})
export class DummyComponent{
 stringToLocalize: string = `@@localizedString`
}
```

Strings that need to be localized must have the '**@@**' prefix which the **flxLocalize** pipe uses to extract and replace the string with a value found in the substitution tags enumeration.

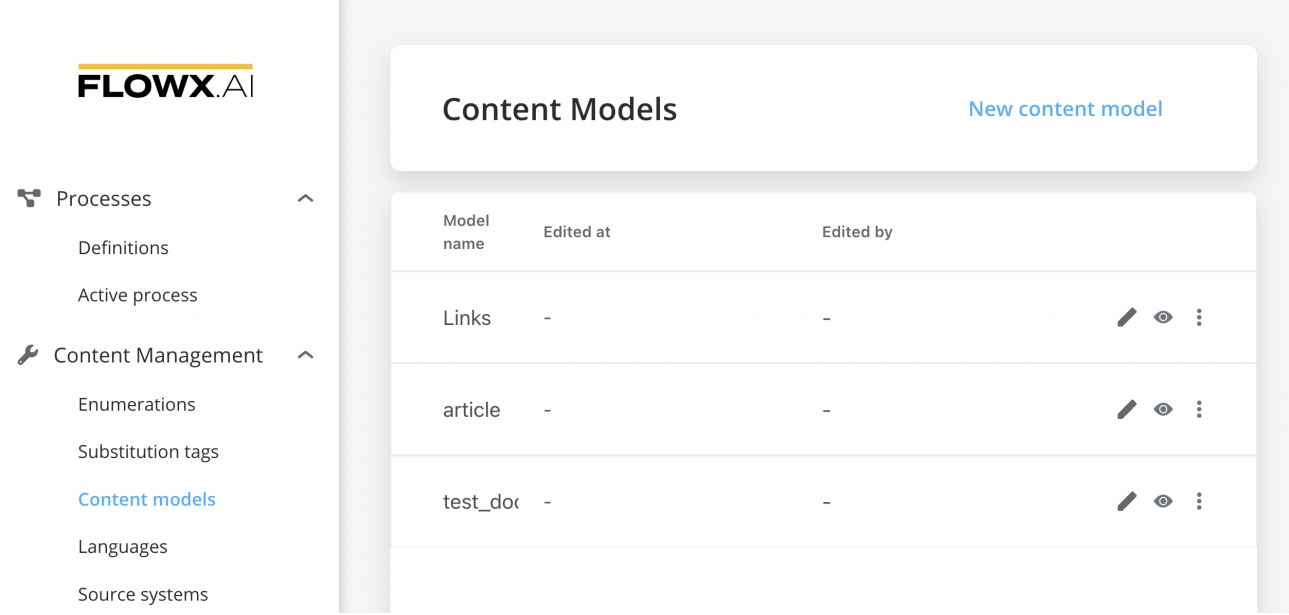
Substitution tags are retrieved when a start process call is first made, and it's cached on subsequent start process calls.

Was this page helpful?

PLATFORM DEEP DIVE / Core components / Core extensions /

Content management / Content models

Content models are used to create complex content collections using customizable pieces of content. For example, you can define content models for an article that will be displayed on a page, creating customizable content.



On the main screen inside **Content models**, you have the following elements:

- **Model name** - the name of the content model
- **Edited at**
- **Edited by**
- **Edit** - button used to edit a content model
- **View process** - click **View process** to open a content model and see its attributes

- **Export/ Delete (Breadcrumbs menu)** - use the breadcrumbs to access the **Export** and **Delete** functions
- **New content model** - button used to add a new content model

For each entry (when you hit the **Open** button) inside an enumeration we have to define the following properties:

- **Language**
- **Name**
- **Attributes (depending on what you add as attributes)**

Configuring attributes

Attributes that can be added to a content model can have the following values:

- **String**
- **Number**
- **Boolean**

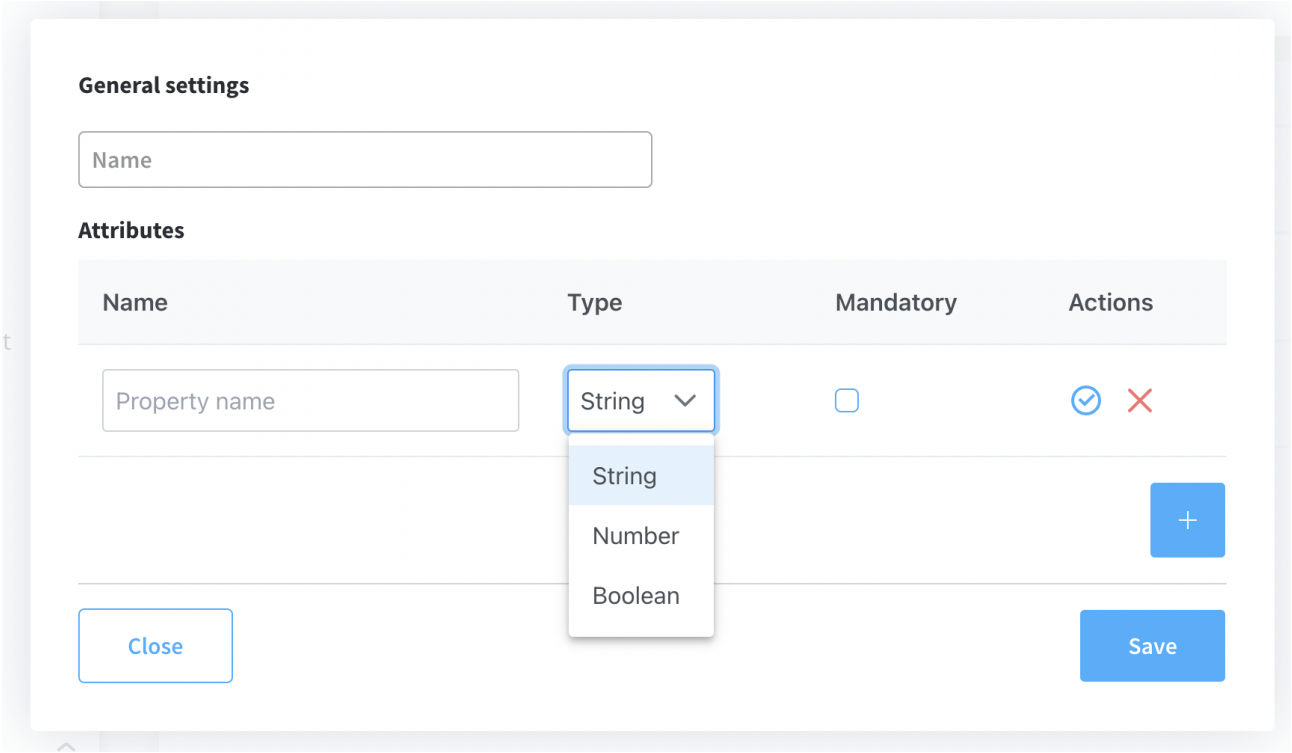
Also, attributes can be either **Mandatory** or **Optional**

Adding new content models

To add a new content model, follow the next steps:

1. Go to **FLOWX Designer** and select the **Content Management** tab.
2. Select **Content models** from the list.
3. Click **New content model**.
4. Add a suggestive name for your content model.

5. Fill in the following details:
    - **Name** (mandatory)
    - **Attributes**:
        - **Name** - add a name for the attribute
        - **Type** - String, Number or Boolean
        - **Mandatory** - mark an Attribute as mandatory
        - **Actions** - add or remove

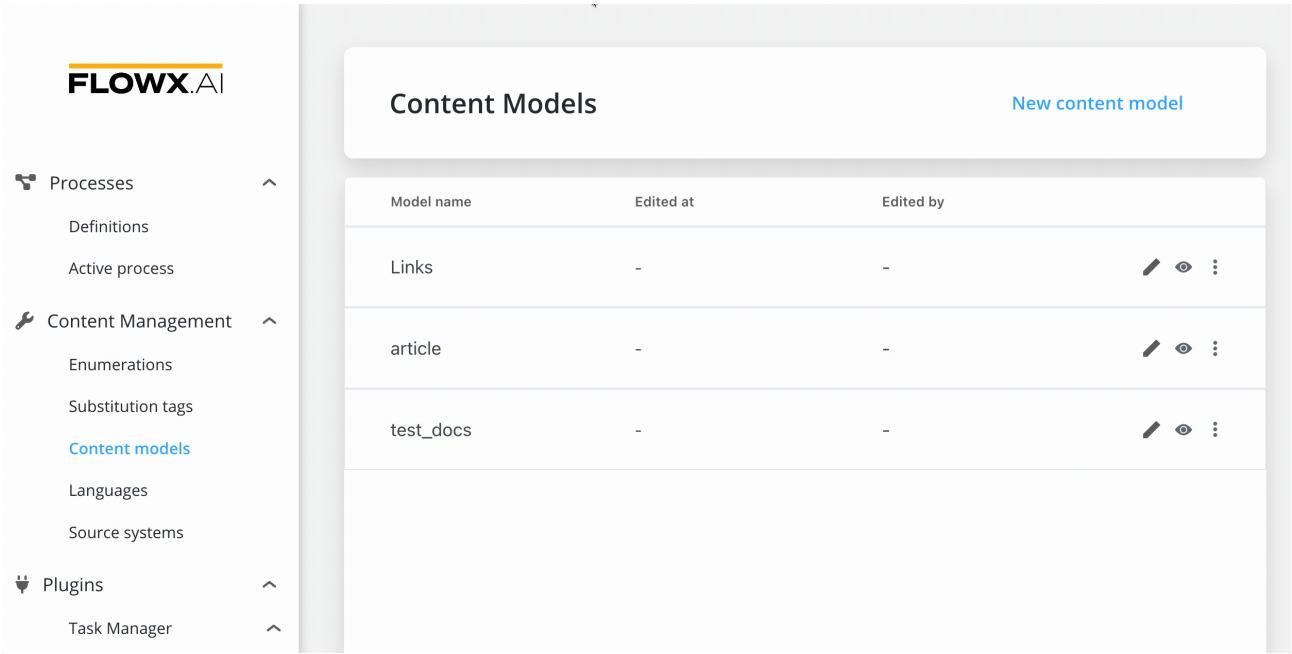


Editing content models

To configure an entry inside a content model, follow the next steps:

1. Go to **FLOWX Designer** and select the **Content Management** tab.
2. Select **Content models** from the list and select a content model.

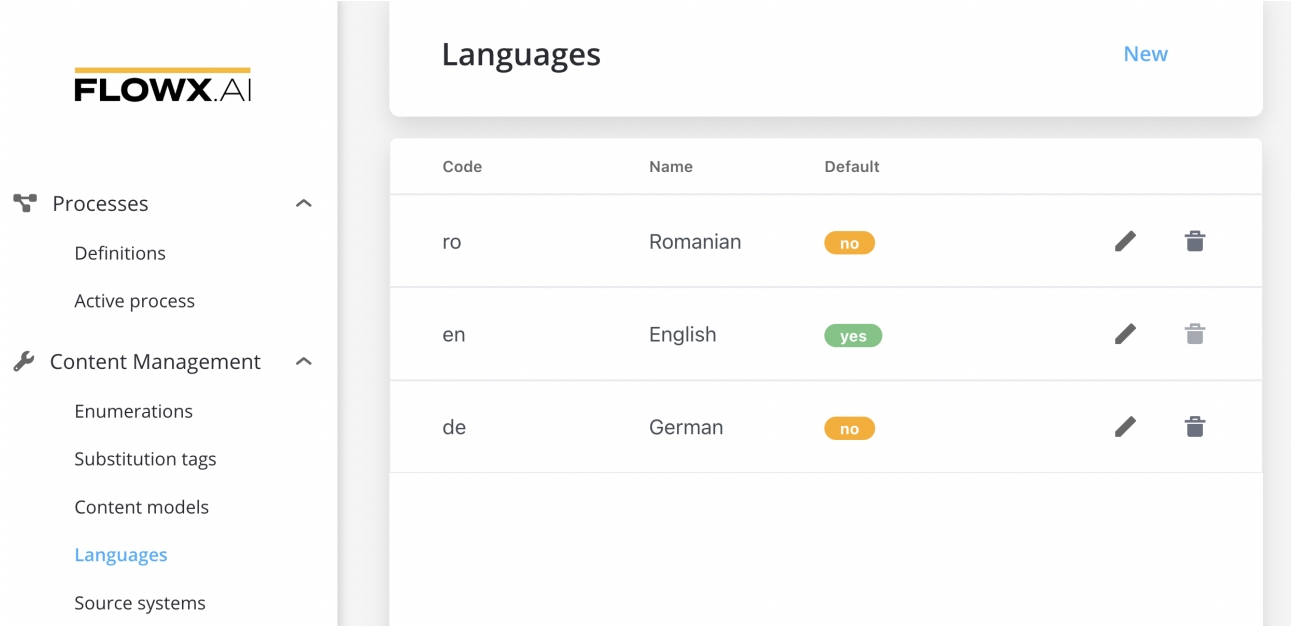
3. Click **View process** and then click **Open**.
4. Click **Edit** to modify the values.



Was this page helpful?

PLATFORM DEEP DIVE / Core components / Core extensions / Content management / Languages

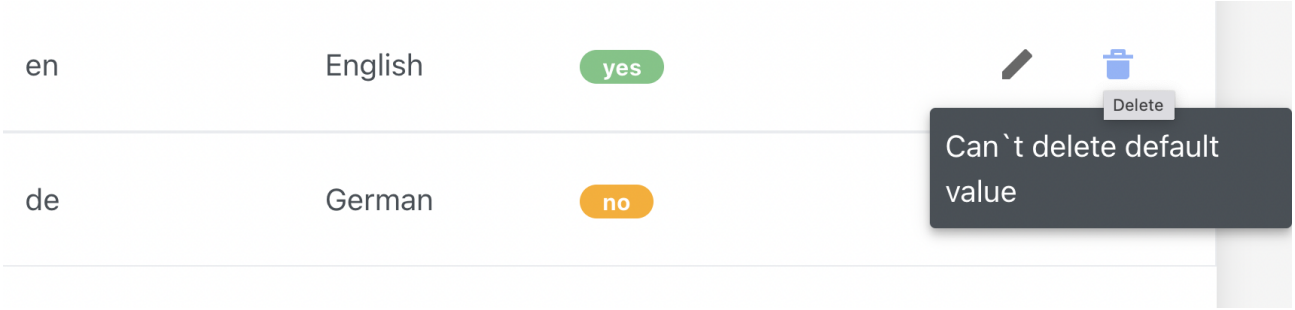
The FLOWX Headless CMS can store and manage languages. You can add a language and use it in almost any content management configuration.



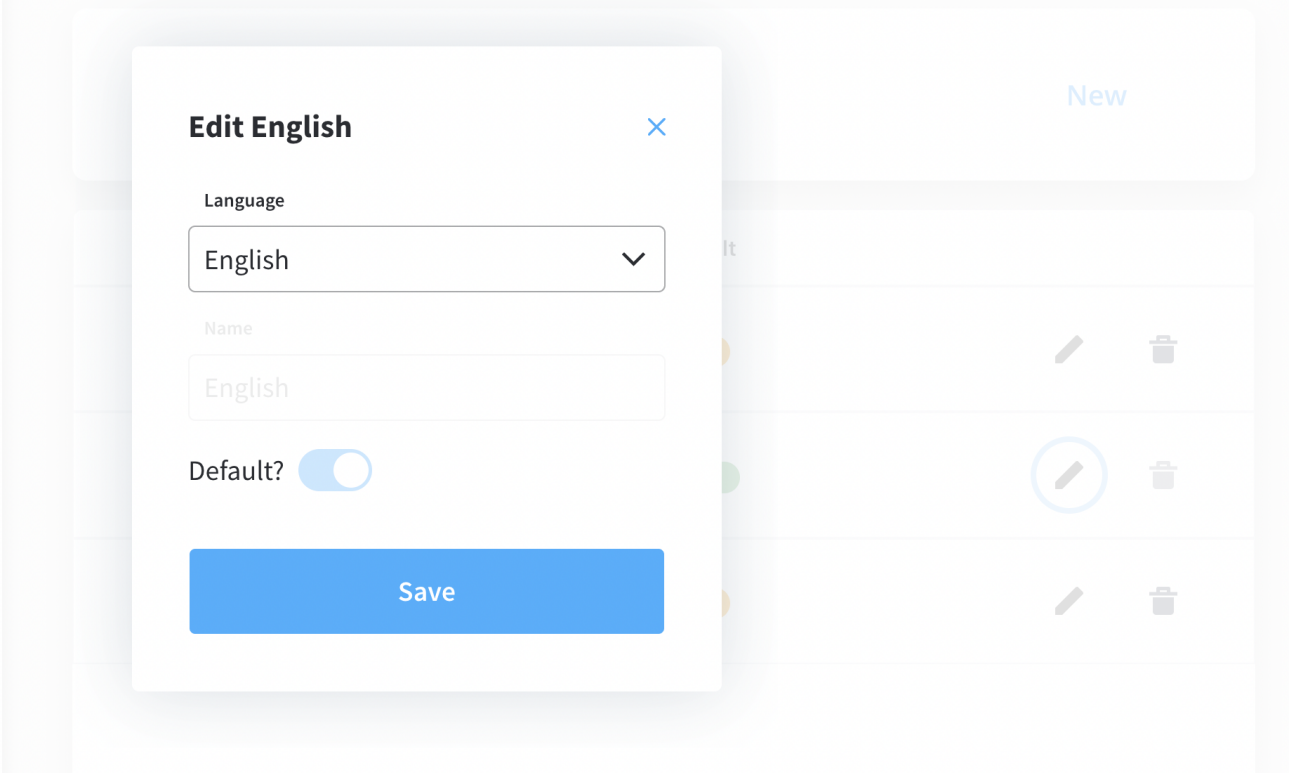
On the main screen inside **Languages**, you have the following elements:

- **Code** - not displayed in the end-user interface, but used to assure value uniqueness
- **Name** - the name of the language
- **Default** - you can set a language as **Default** (default values can't be deleted)

> **CAUTION**
>
> When working with substitution tags or other elements that imply values from other languages defined in the CMS, when running a process, the default values extracted will be the ones marked by the default language.



- **Edit** - button used to edit a language



- **Delete** - button used to delete a language

DANGER

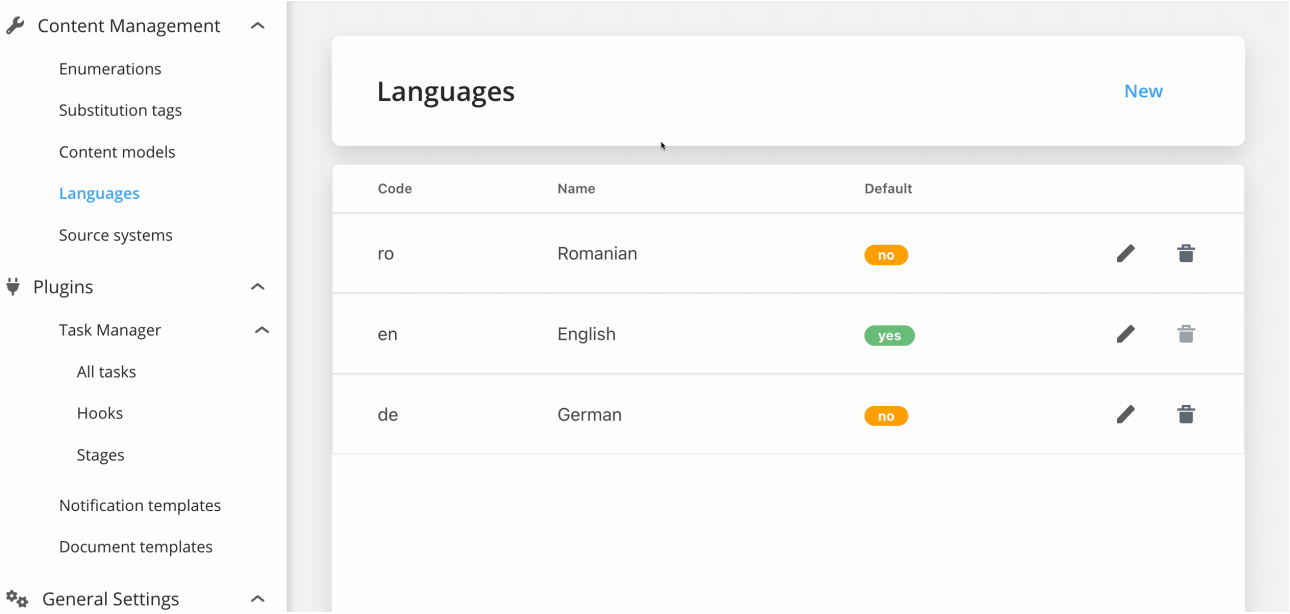
Before deleting a language make sure that this is not used in any content management configuration.

- **New** - button used to add a new language

Adding a new language

To add a new language, follow the next steps:

1. Go to
   The fallback content to display on prerendering
   and select the **Content Management** tab.
2. Select **Languages** from the list.
3. Choose a new **language** from the list.
4. Click **Add** after you finish.

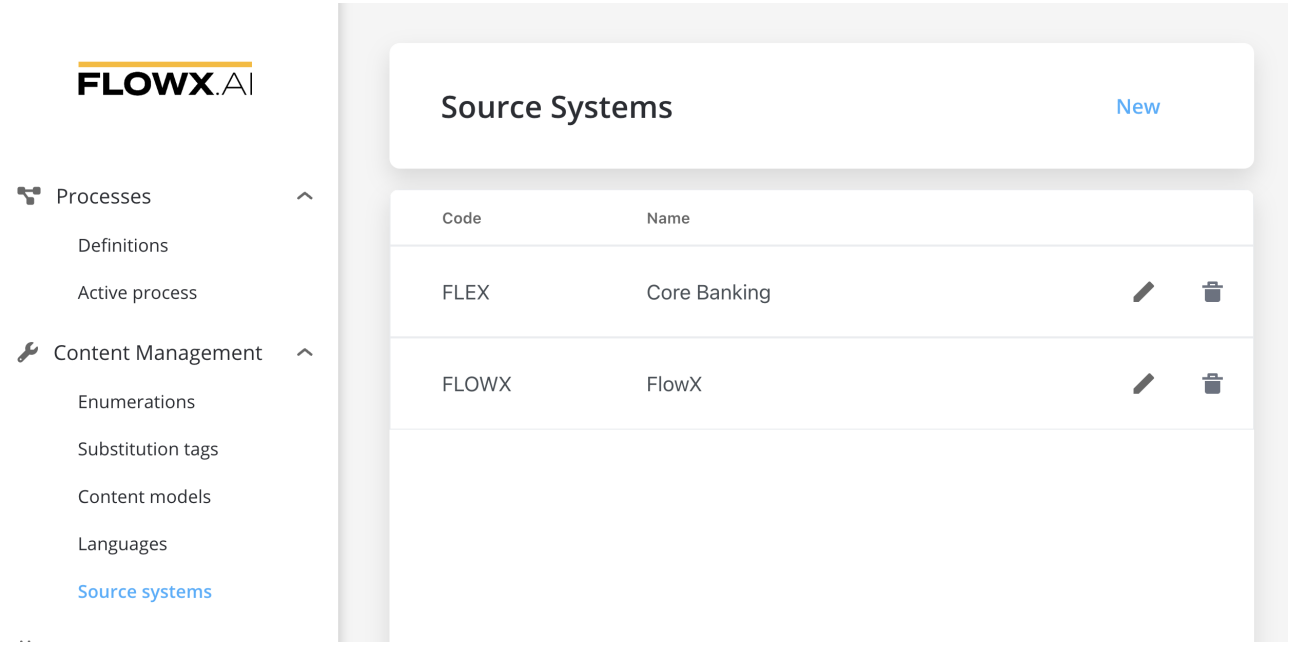


Was this page helpful?

PLATFORM DEEP DIVE / Core components / Core extensions / Content management / Source systems

If multiple

The fallback content to display on prerendering
values are needed to communicate with other systems, source systems can be used.



On the main screen inside **Source systems**, you have the following elements:

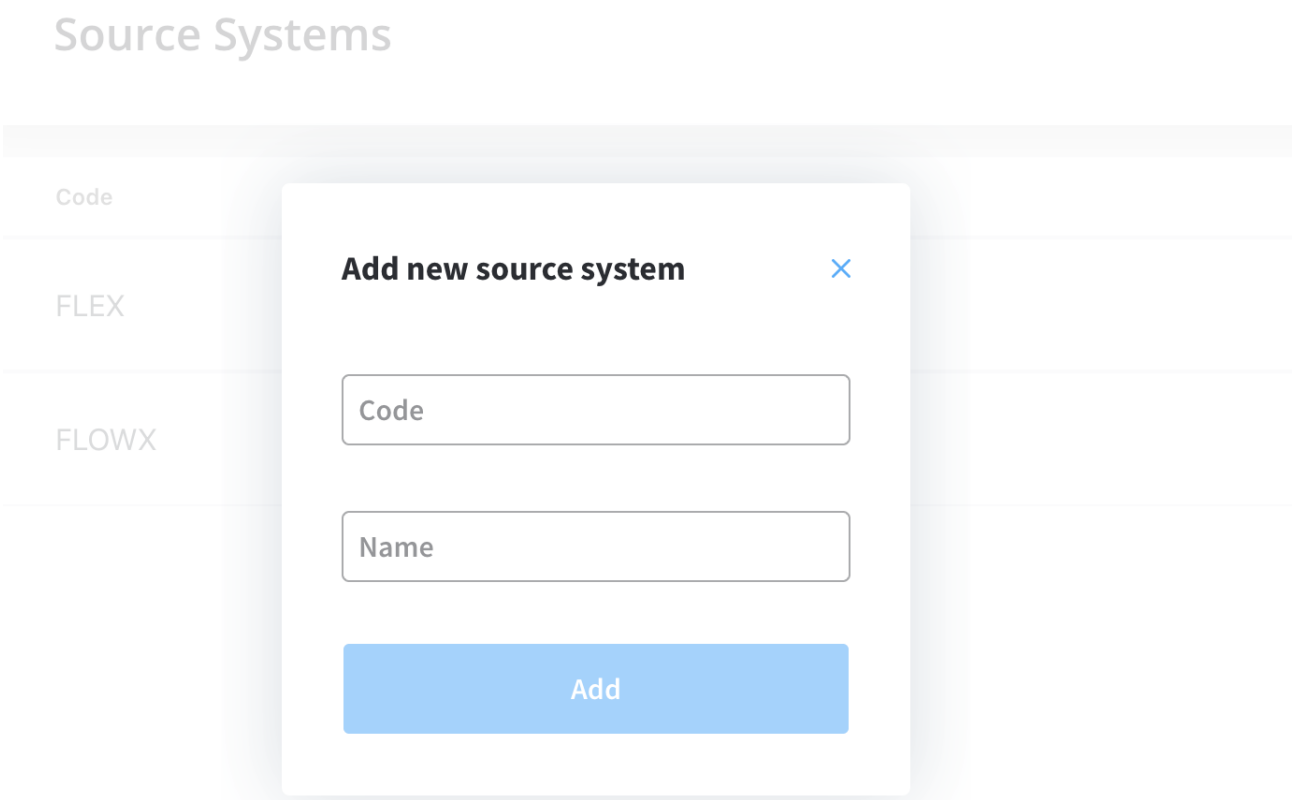
- **Code** - not displayed in the end-user interface, but used to assure value uniqueness

- **Name** - the name of the source system
- **Edit** - button used to edit a source system
- **Delete** - button used to delete a source system
- **New** - button used to add a new source system

Adding new source systems

To add a new source system, follow the next steps.

1. Go to
   The fallback content to display on prerendering
   and select the **Content Management** tab.
2. Select **Source systems** from the list.
3. Fill in the necessary details:
   - Code
   - Name
4. Click **Add** after you finish.

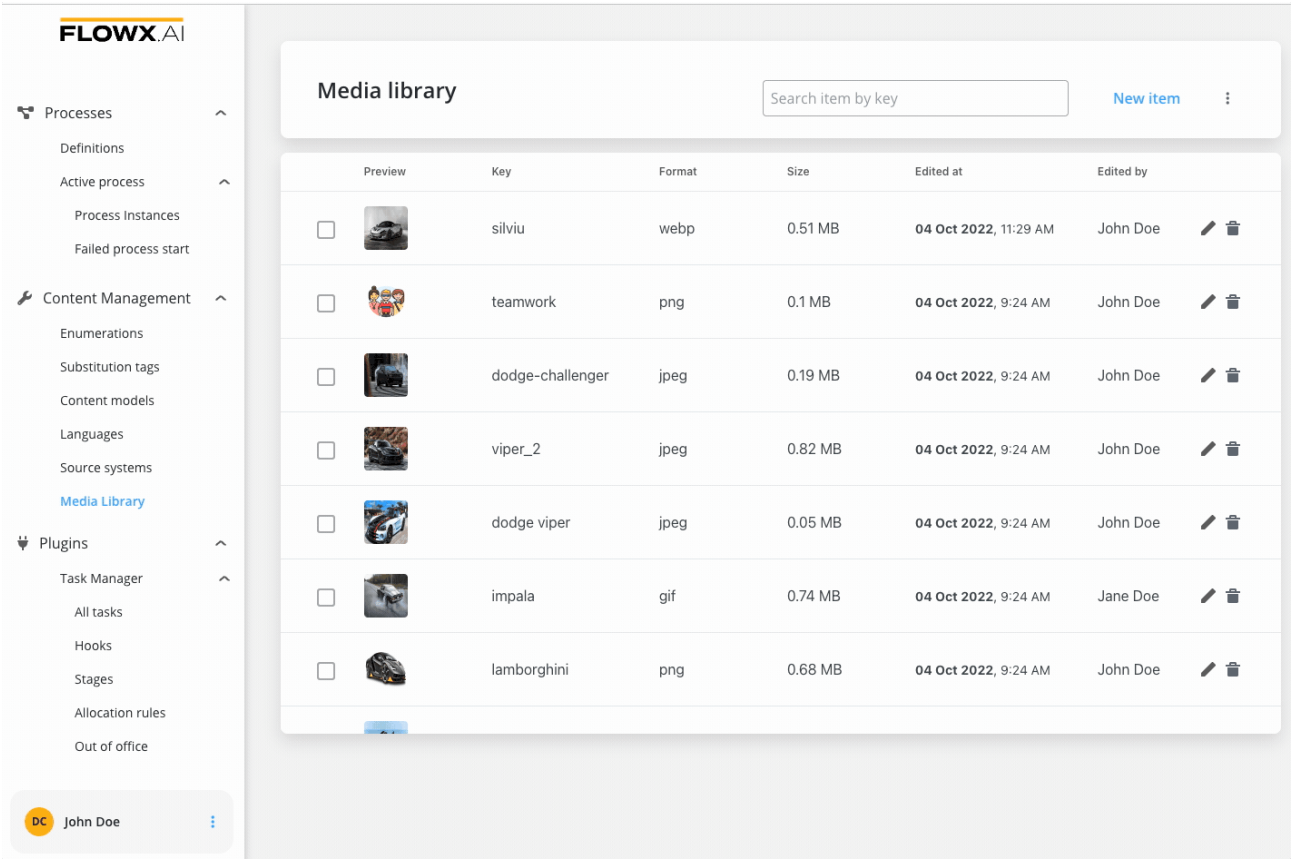


Was this page helpful?

PLATFORM DEEP DIVE / Core components / Core extensions / Content management / Media library

The media library serves as a centralized hub for managing and organizing various types of media files, including images, GIFs, and more. It encompasses all the files that have been uploaded to the

The fallback content to display on prerendering
, providing a convenient location to view, organize, and upload new media files.



INFO

You can also upload an image directly to the Media Library on the spot when configuring a process using the **UI Designer**. More information **here**.

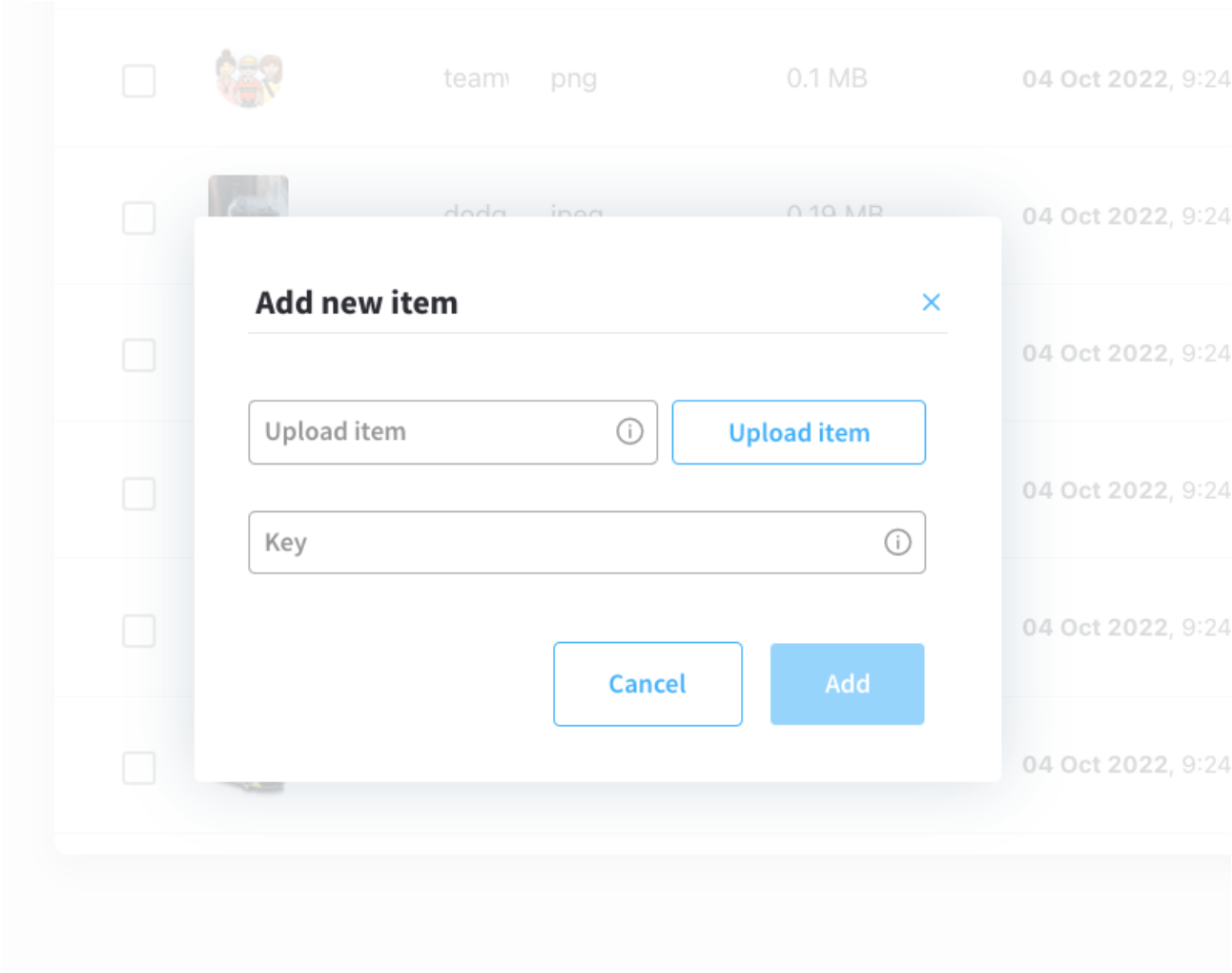
Uploading a new asset

To upload an asset to the Media Library, follow the next steps:

1. Open

   The fallback content to display on prerendering

   .
2. Go to **Content Management** tab and select **Media Library**.
3. Click **Add new item**, the following details will be displayed:
   - **Upload item** - opens a local file browser
   - **Key** - the key must be unique, you cannot change it afterwards



4. Click **Upload item** button and select a file from your local browser.
5. Click **Upload item** button again to upload the asset.

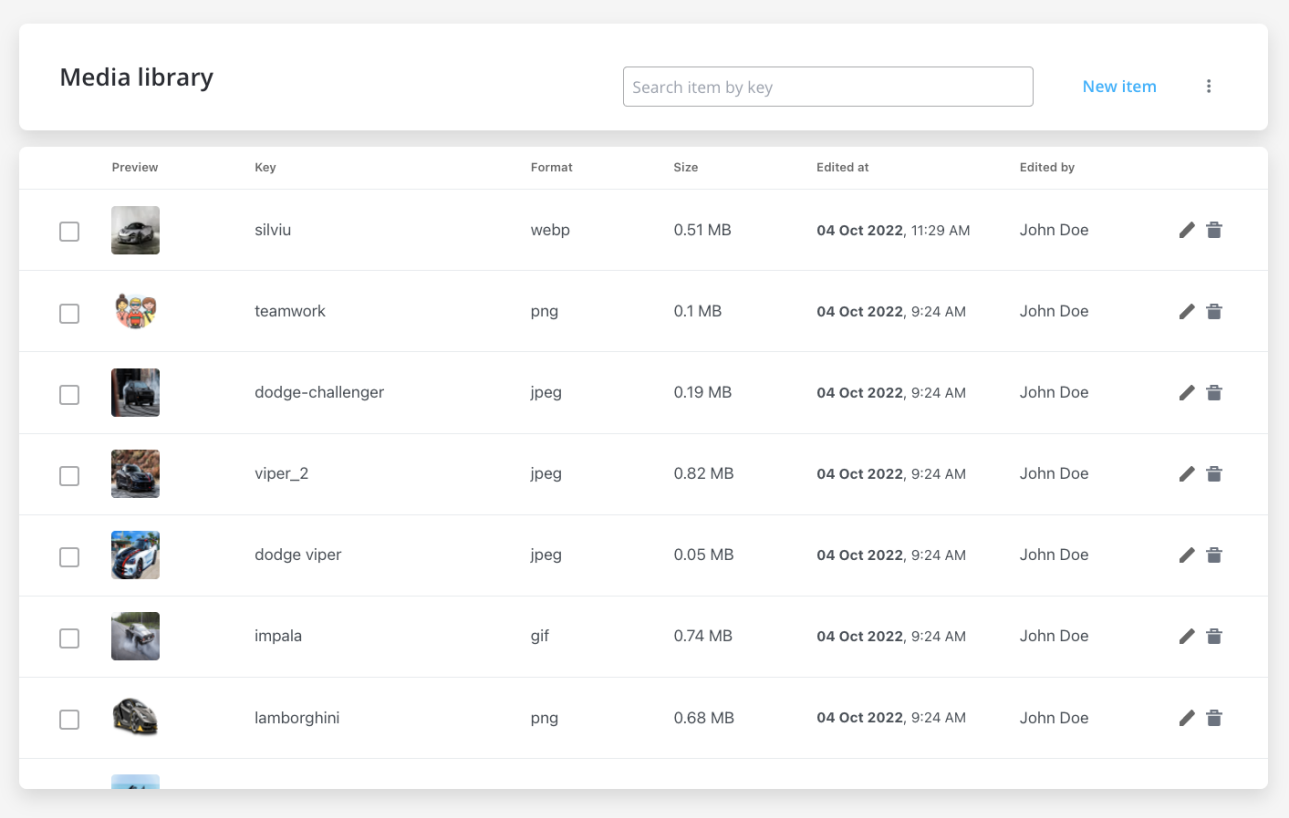
> **CAUTION**
>
> Supported formats: PNG, JPEG, JPG, GIF, SVG or WebP format, 1 MB maximum size.

Displaying assets

Users can preview all the uploaded assets just be accessing the **Media Library**.

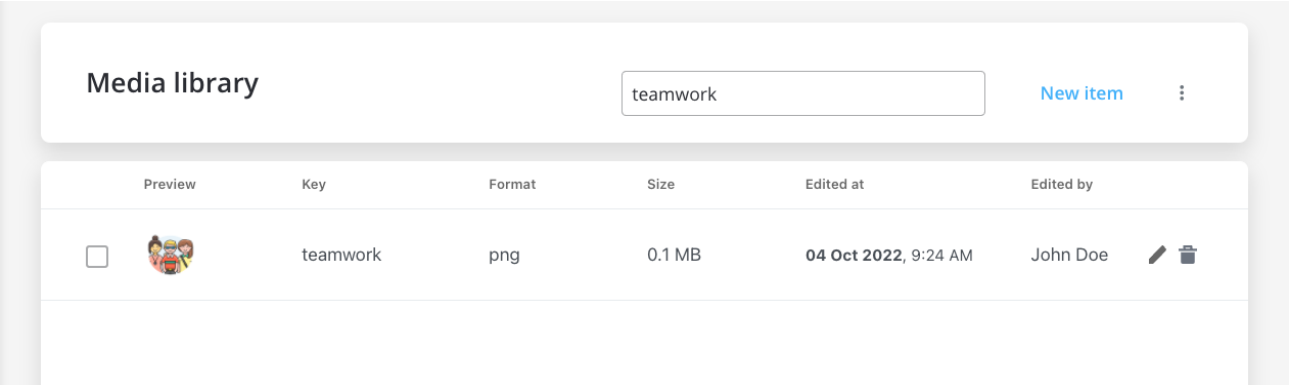
You have the following information about assets:

- Preview (thumbnail 48x48)
- Key
- Format ("-" for unknown format)
- Size
- Edited at
- Edited by



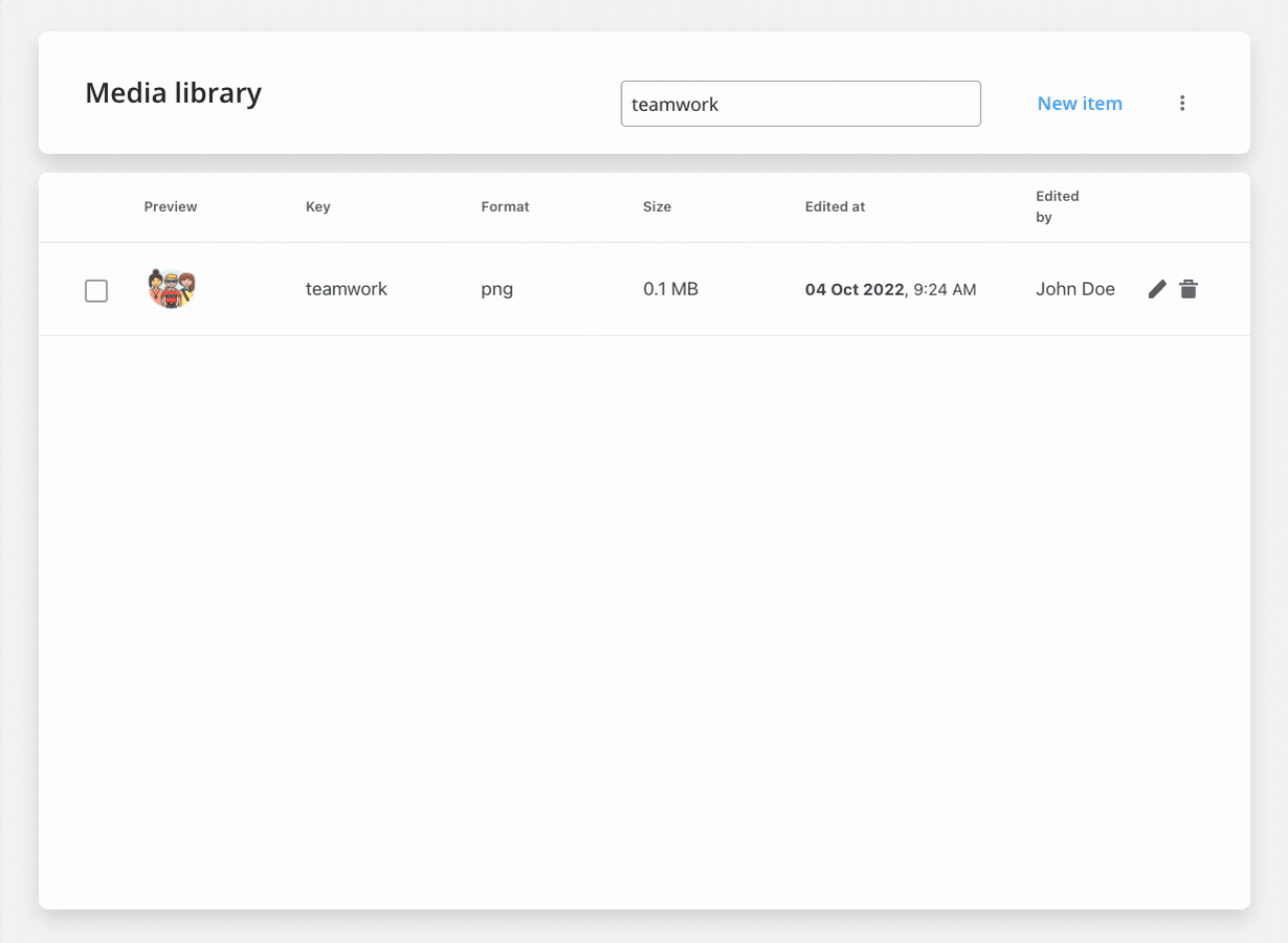
Searching assets

You can search an asset by using its key (full or substring).



Replacing assets

You can replace an item on a specific key (this will not break references to process definitions).

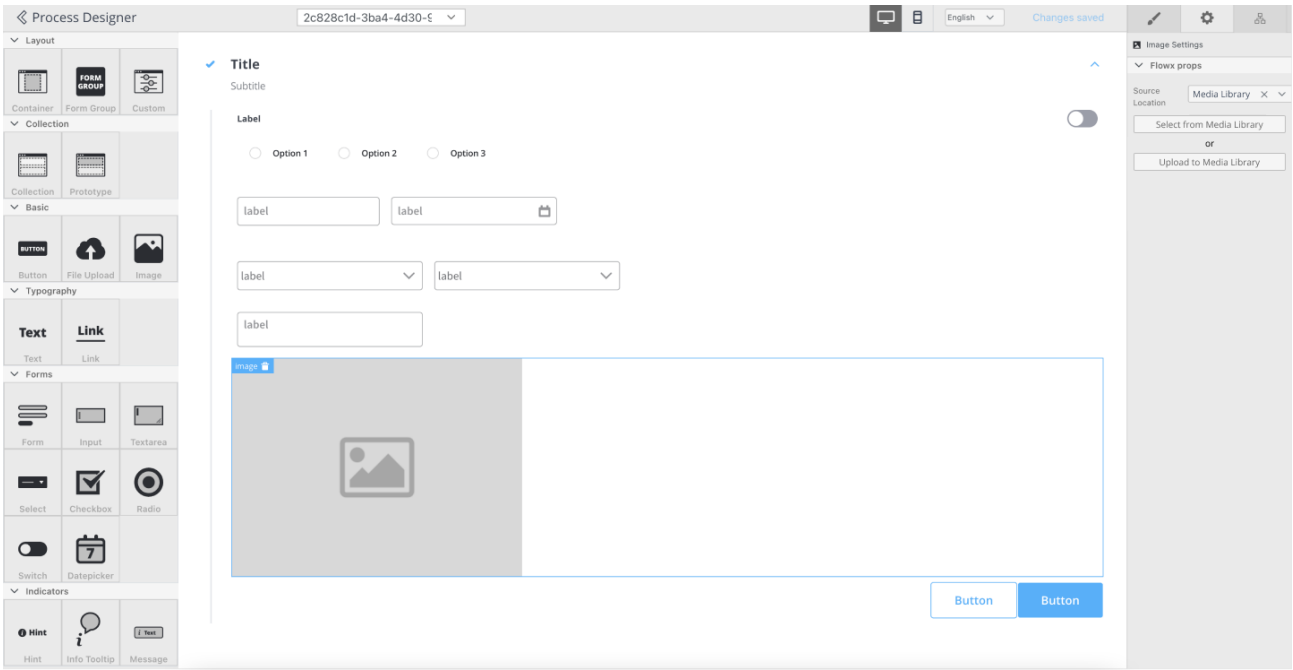


Referencing assets in UI Designer

You have the following options when configuring image components using UI Designer:

- Source Location - here you must select **Media Library** as source location
- Image Key

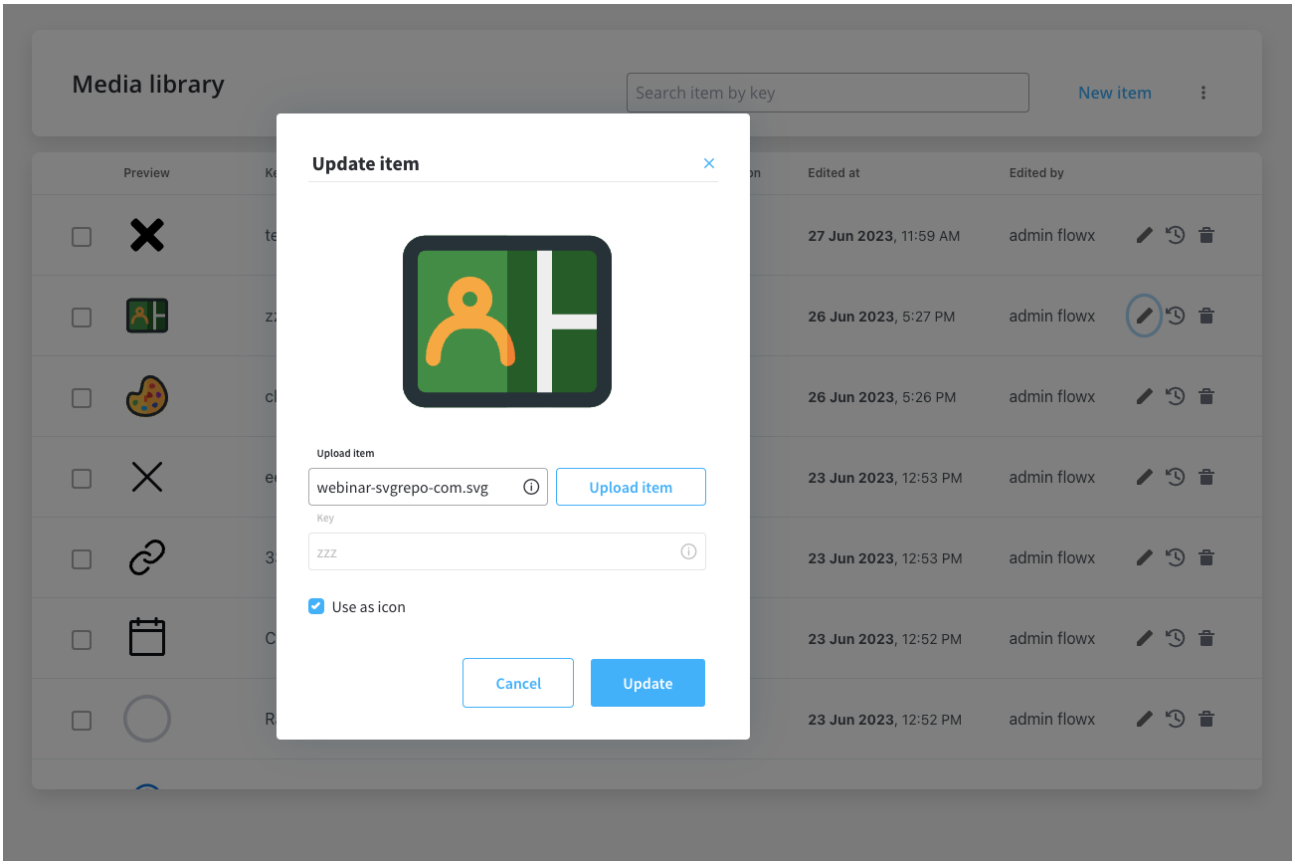
- **Option 1**: trigger a dropdown with images keys - you can type and filter options or can select from the initial list in dropdown
- **Option 2**: open a popup with images thumbnails and keys then you can type and filter options or can select from the initial list



> **INFO**
>
> More details on how to configure an image component using UI Designer - **here**.

Icons

The Icons feature allows you to personalize the icons used in UI elements. By uploading SVG files through the Media Library and marking them, you can choose icons from the available list in the UI Designer.



INFO

When selecting icons in the UI Designer, only SVG files marked as icons in the Media Library will be displayed.

INFO

To ensure optimal visual rendering and alignment within your UI elements, it is recommended to use icons with small sizes such as: 16px, 24px, 32px.

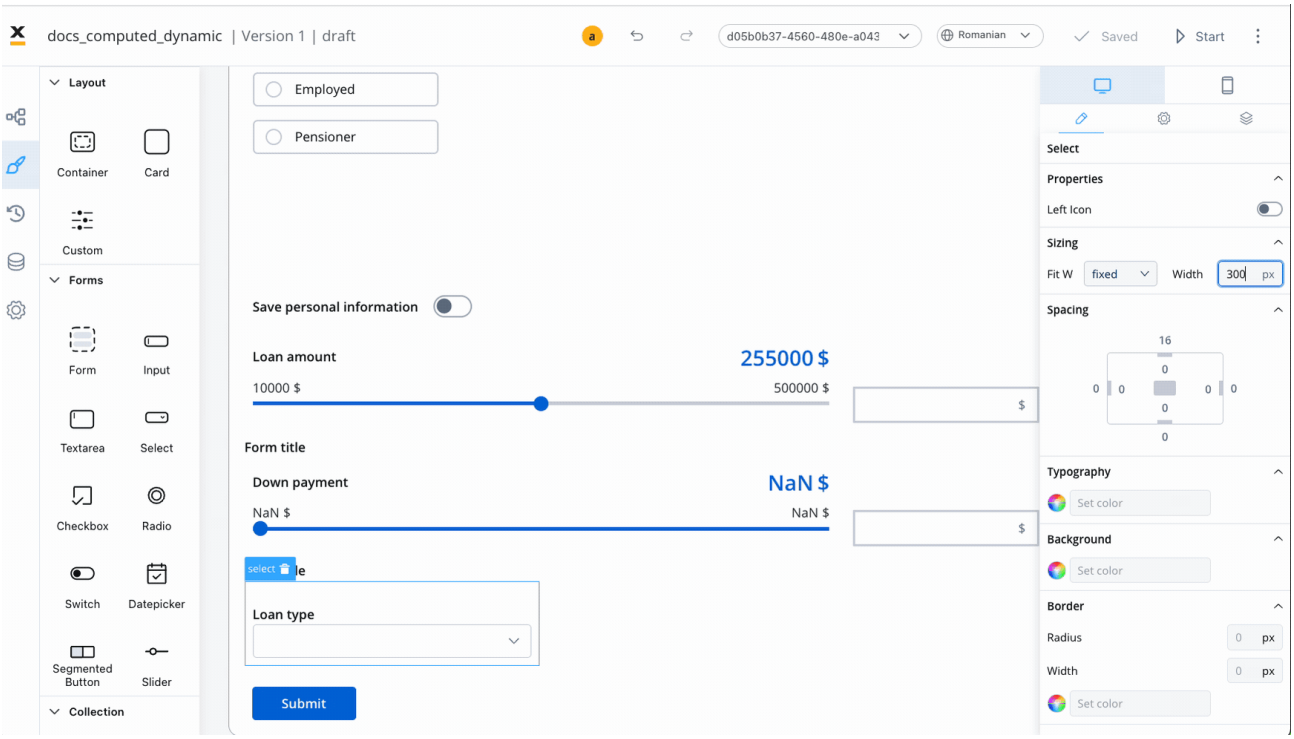
Using icons specifically designed for these sizes helps maintain consistency and ensures a visually pleasing user interface. It is advisable to select icons

from icon sets that provide these size options or to resize icons proportionally to fit within these dimensions.

> **CAUTION**
>
> Icons are displayed or rendered at their original, inherent size.

Customization

Content-specific icons pertain to the content of UI elements, such as icons for input fields or send message buttons. These icons are readily accessible in the UI Designer.



More details on how to add icons on each element, check the sections below:

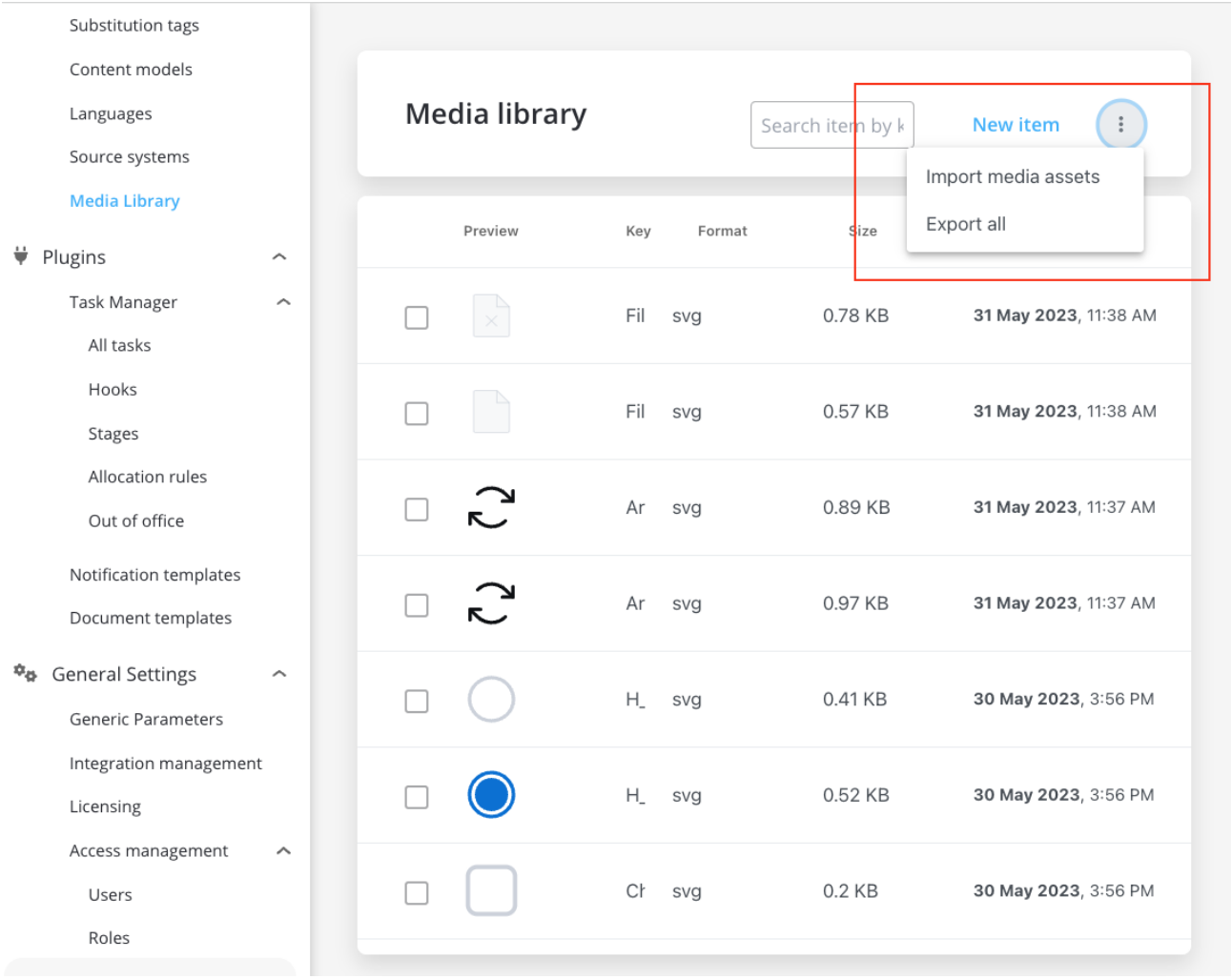
» Input element

» Select element

» Buttons

Export/import media assets

The import/export feature allows you to import or export media assets, enabling easy transfer and management of supported types of media files.



Import media assets

Use this function to import media assets of various supported types. It provides a convenient way to bring in images, videos, or other media resources.

Export all

Use this function to export all media assets stored in your application or system. The exported data will be in JSON format, allowing for easy sharing, backup, or migration of the media assets.

The exported JSON structure will resemble the following example:


```
{
  "images": [
    {
      "key": "cart",
      "application": "flowx",
      "filename": "maxresdefault.jpg",
      "format": "jpeg",
      "contentType": "image/jpeg",
      "size": 39593,
      "storagePath":
"https://d22tnnndi9lo60.cloudfront.net/devmain/flowx/cart/1681
      "thumbnailStoragePath":
"https://d22tnnndi9lo60.cloudfront.net/devmain/flowx/cart/1681
    },
    {
      "key": "pizza",
      "application": "flowx",
      "filename": "pizza.jpeg",
      "format": "jpeg",
      "contentType": "image/jpeg",
      "size": 22845,
      "storagePath":
"https://d22tnnndi9lo60.cloudfront.net/devmain/flowx/pizza/168
      "thumbnailStoragePath":
"https://d22tnnndi9lo60.cloudfront.net/devmain/flowx/pizza/168
    }
  ],
  "exportVersion": 1
}
```

- `images`- is an array that contains multiple objects, each representing an image
- `exportVersion` - represents the version number of the exported data, it holds the image-related information
- `key`- represents a unique identifier or name for the image, it helps identify and differentiate images within the context of the application
- `application` - specifies the name or identifier of the application associated with the image, it indicates which application or system the image is related to
- `filename` - the name of the file for the image, it represents the original filename of the image file
- `format` - a string property that specifies the format or file extension of the image
- `contentType` - the MIME type or content type of the image, it specifies the type of data contained within the image file
- `size` - represents the size of the image file in bytes, it indicates the file's storage size on a disk or in a data storage system
- `storagePath` - the URL or path to the location where the original image file is stored, it points to the location from where the image can be accessed or retrieved
- `thumbnailStoragePath` - the URL or path to the location where a thumbnail version of the image is stored, it points to the location from where the thumbnail image can be accessed or retrieved

Was this page helpful?