# ELO Flows Development

Flow component

# Table of contents

# Flow component

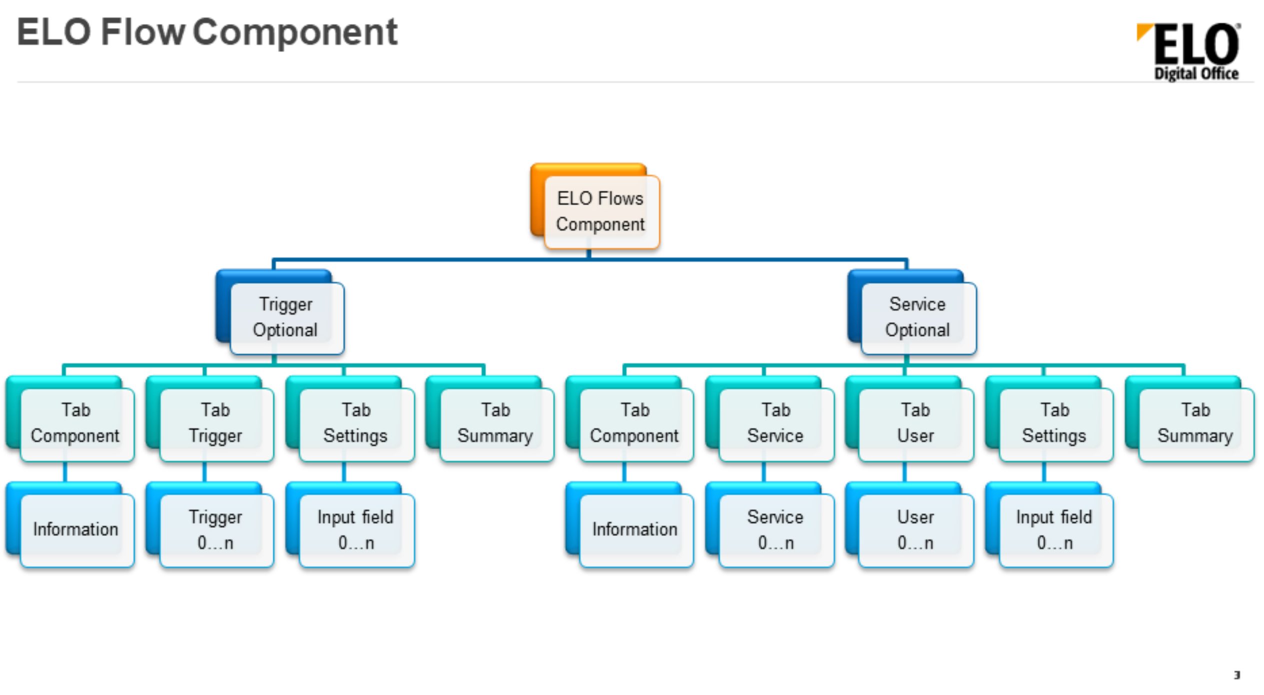
## Component structure

The ELO Flows component can initially provide one or more triggers, one or more services, or both. All triggers and services can be configured. Data can be exchanged between the components via keys. Each component provides extensive handling information for its triggers and services.

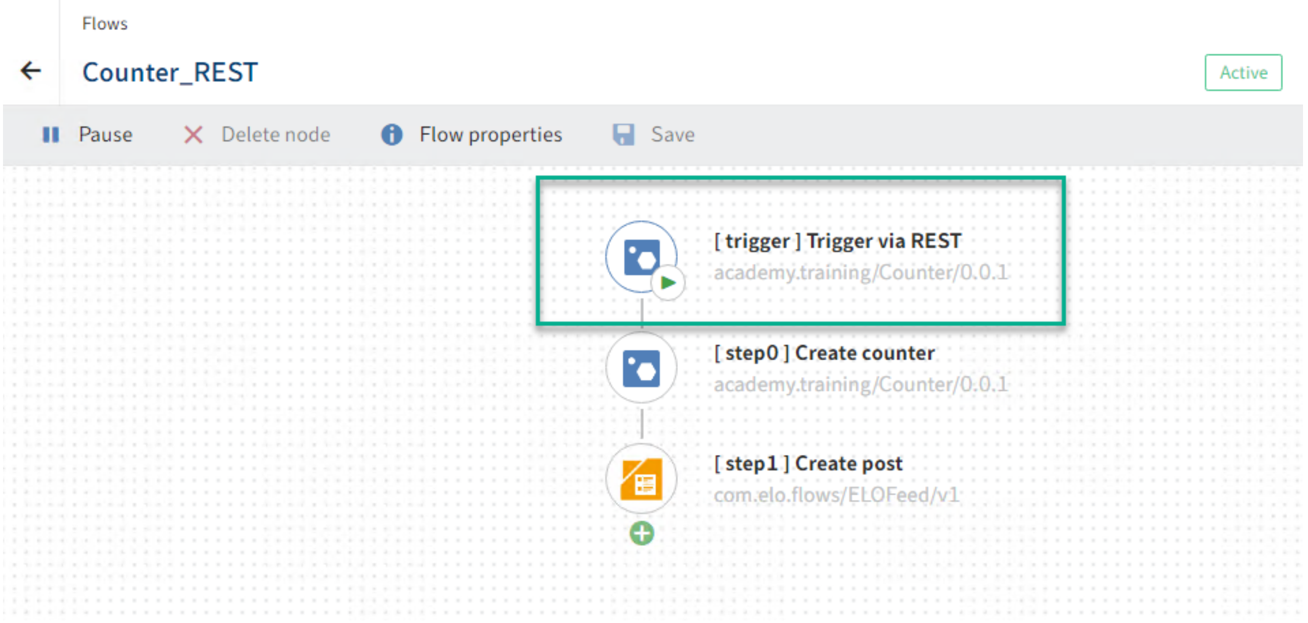
The component is configured in a graphical user interface. Custom settings can be made here. It is programmed in Java using annotations in particular.

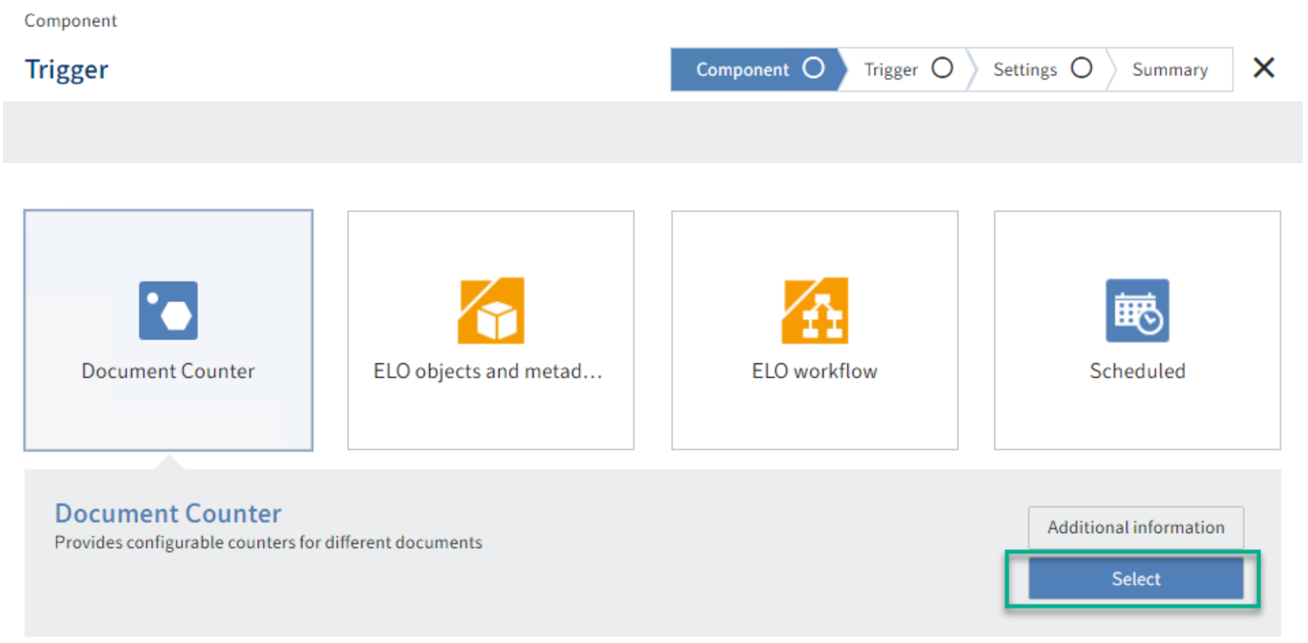
Data is exchanged via an internal JSON configuration object. The data is implemented in input and output classes.

The function descriptions of the components are provided by integrating *markdown files (.md)*.

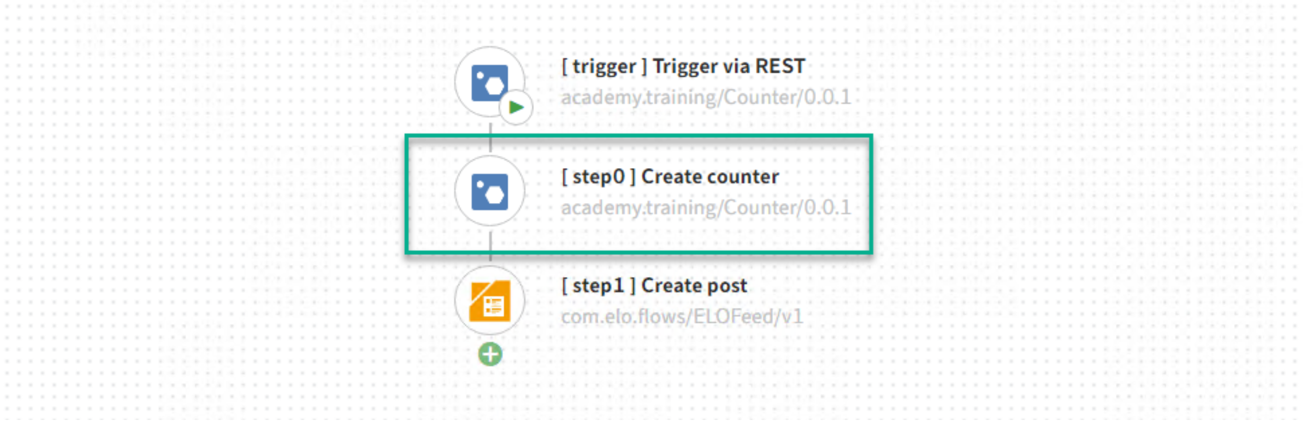


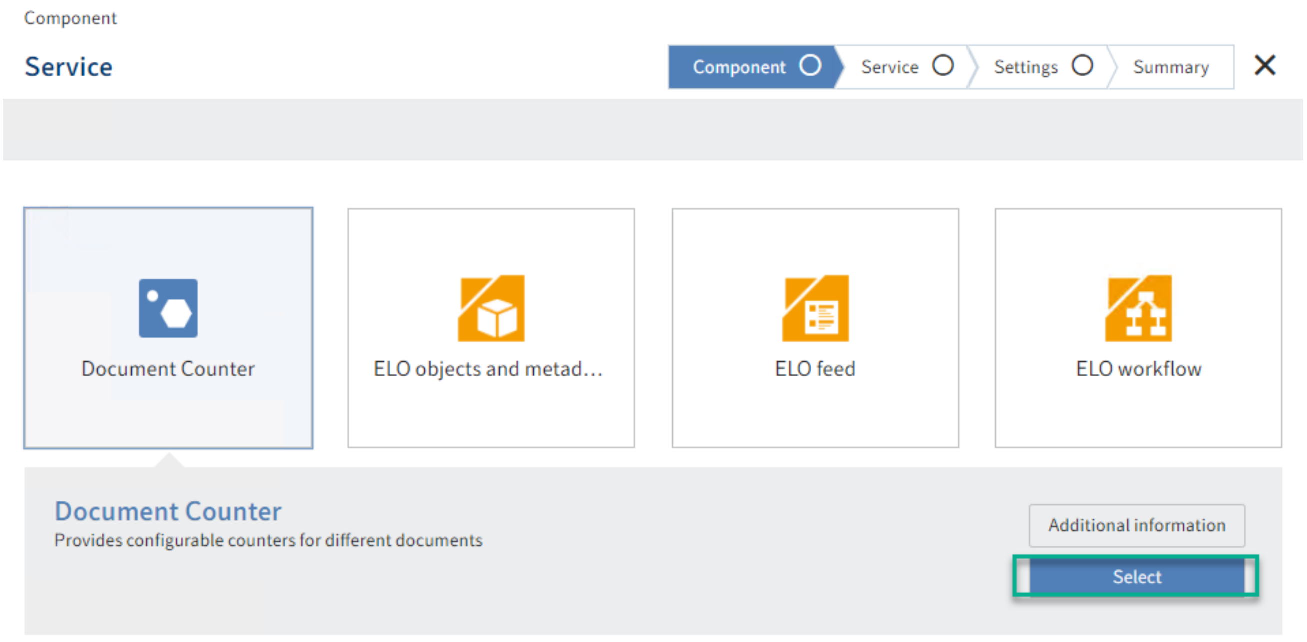
Trigger implementation.





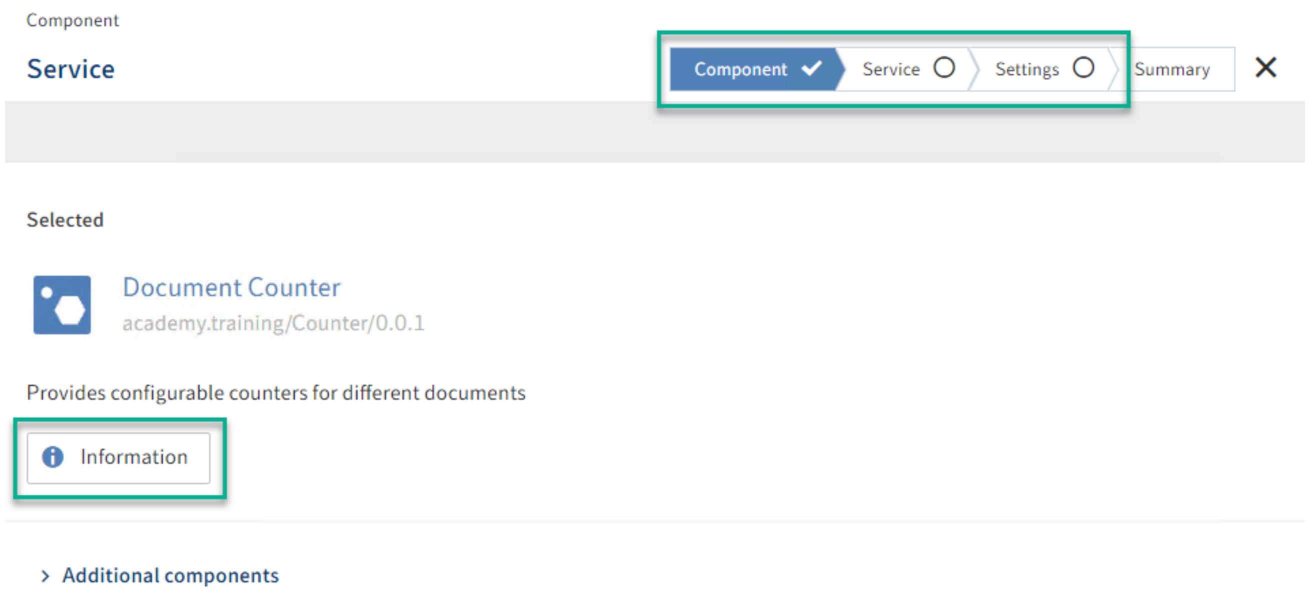
Service implementation.


[ trigger ] Trigger via REST
academy.training/Counter/0.0.1
[ step0 ] Create counter
academy.training/Counter/0.0.1
[ step1 ] Create post
com.elo.flows/ELOFeed/v1



Provides configurable counters for different documents

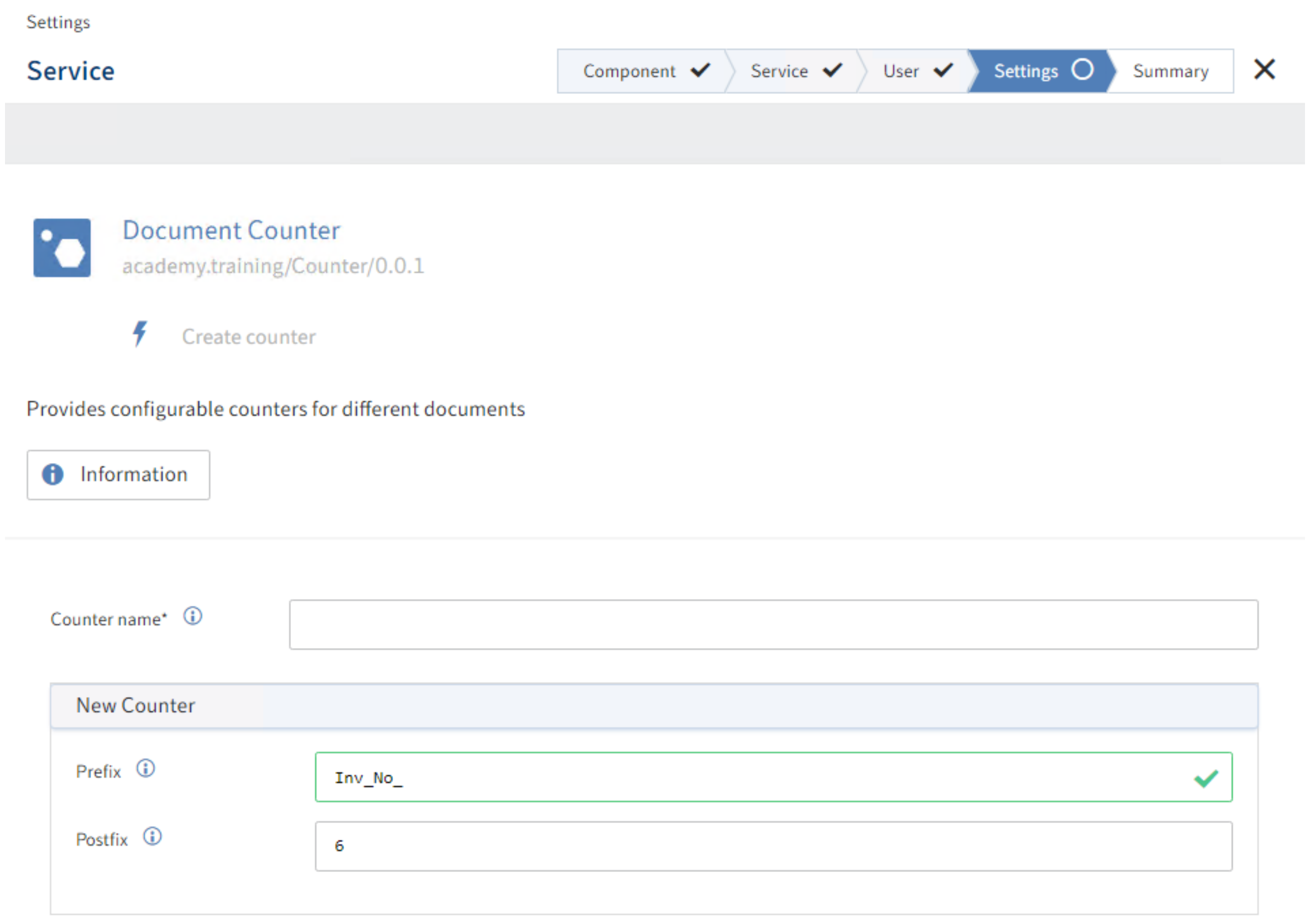
Implementation options within a component.



Document Counter
academy.training/Counter/0.0.1

Provides configurable counters for different documents

Implementation options within the Settings tab.



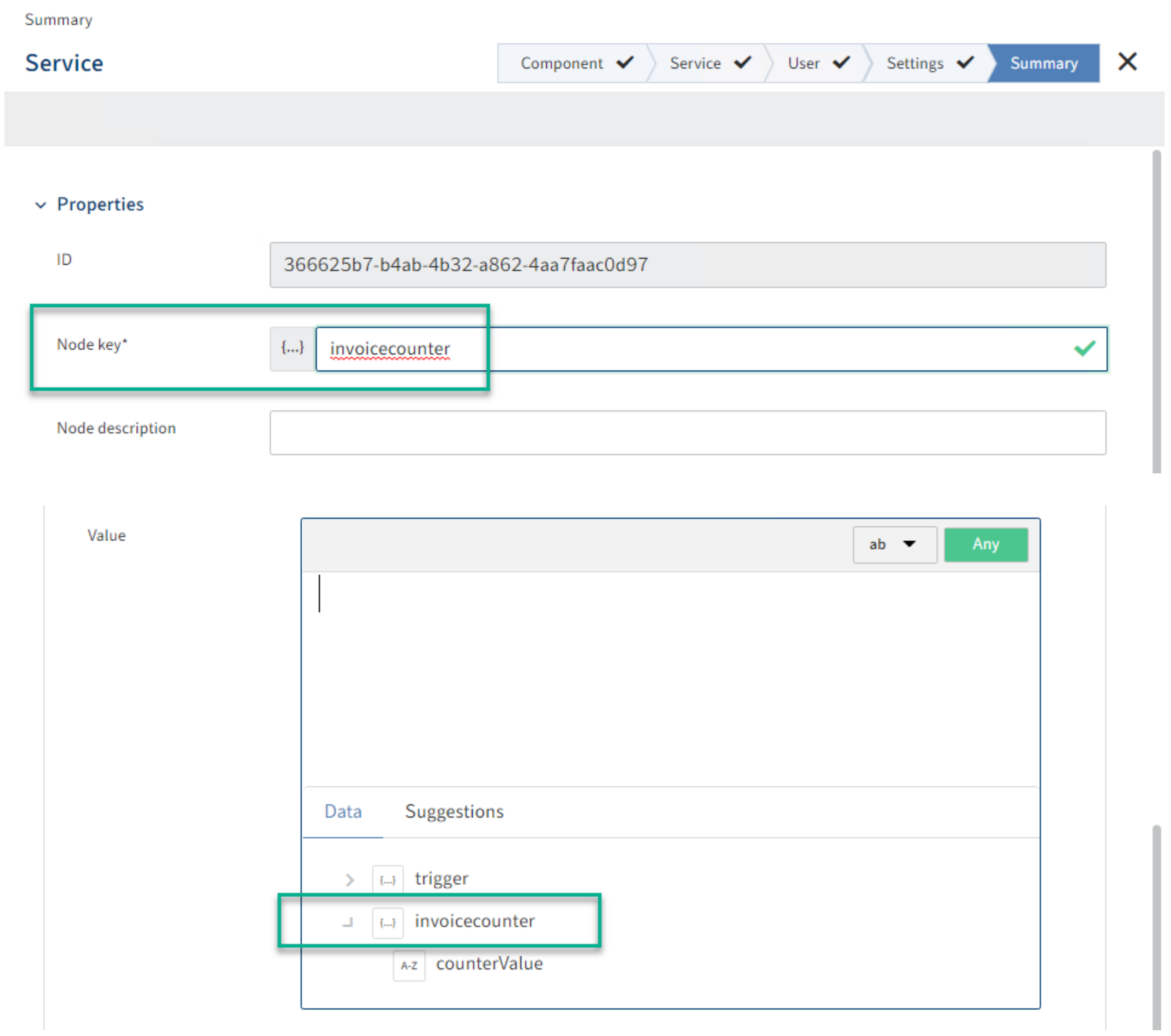
Document Counter
academy.training/Counter/0.0.1

Create counter

Provides configurable counters for different documents

| | |
|---|---|
| Counter name* | |
| **New Counter** | |
| Prefix | Inv_No_ |
| Postfix | 6 |

Data can be forwarded to the following components in a JSON configuration object via keys.


Summary

Service

Properties

ID: 366625b7-b4ab-4b32-a862-4aa7faac0d97

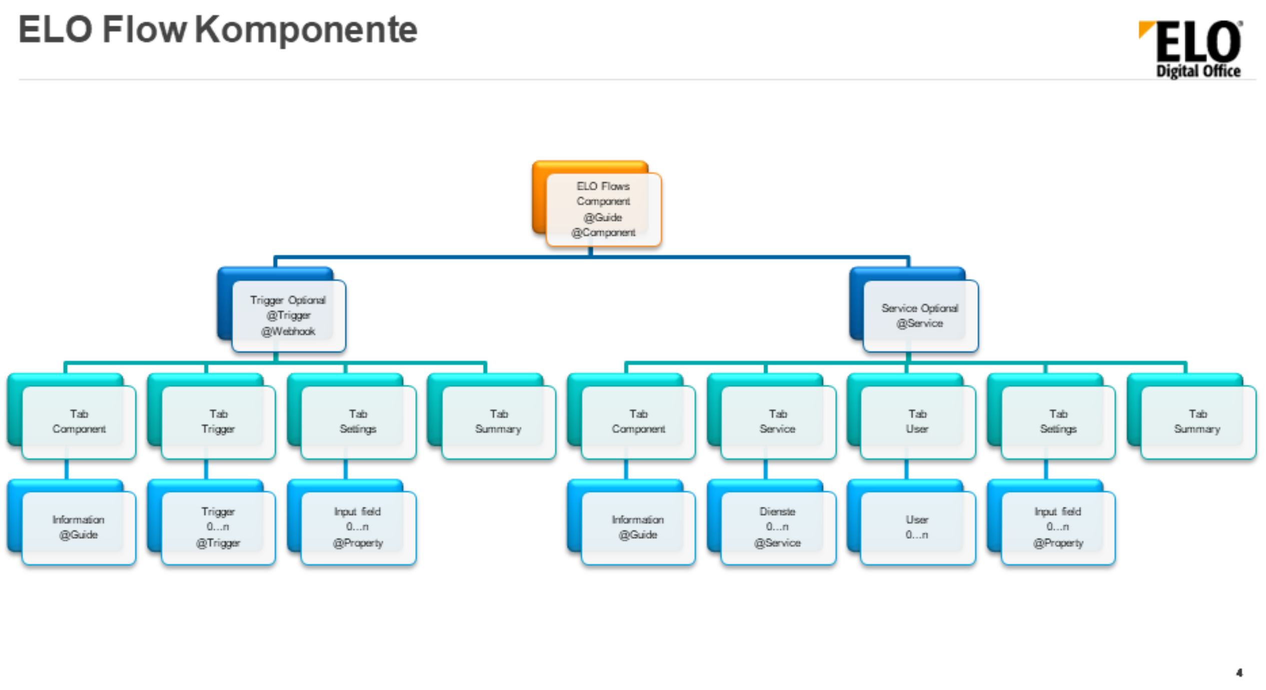
Node key*: invoicecounter

Node description:

Value

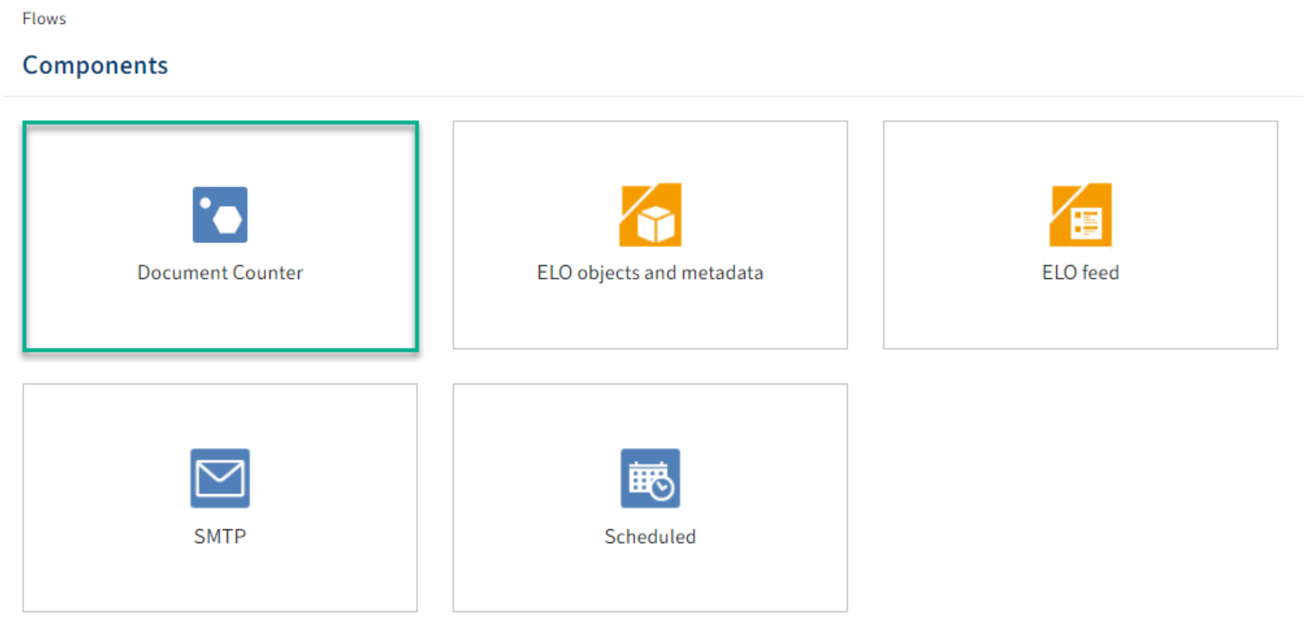
## Overview of ELO Flows components

Below, all graphical components are addressed in examples with their corresponding annotations.



### ELO Flows component

Each flow is based on components. Components can contain multiple elements. For example, a component provides one or more triggers. These triggers then initiate additional actions in the flow. These actions are realized via services. Services are also provided by components.



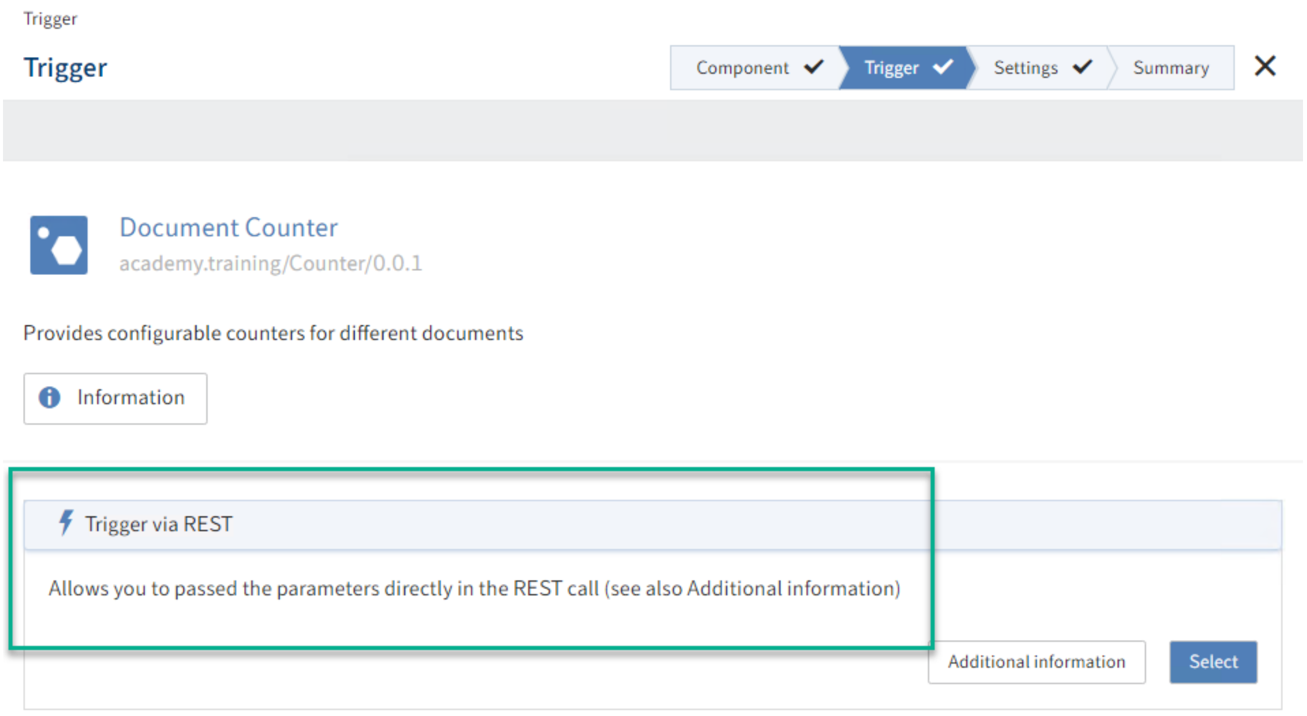
Definition of the component class (here: CounterComponent):

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName = "Do
public class CounterComponent {
}
```

## Triggers

Triggers are used to call a flow.

If a component in a flow is used as the first component, it automatically provides triggers in the ELO Flows administration area.


Trigger

Trigger

Document Counter
academy.training/Counter/0.0.1

Provides configurable counters for different documents

Trigger via REST

Allows you to passed the parameters directly in the REST call (see also Additional information)

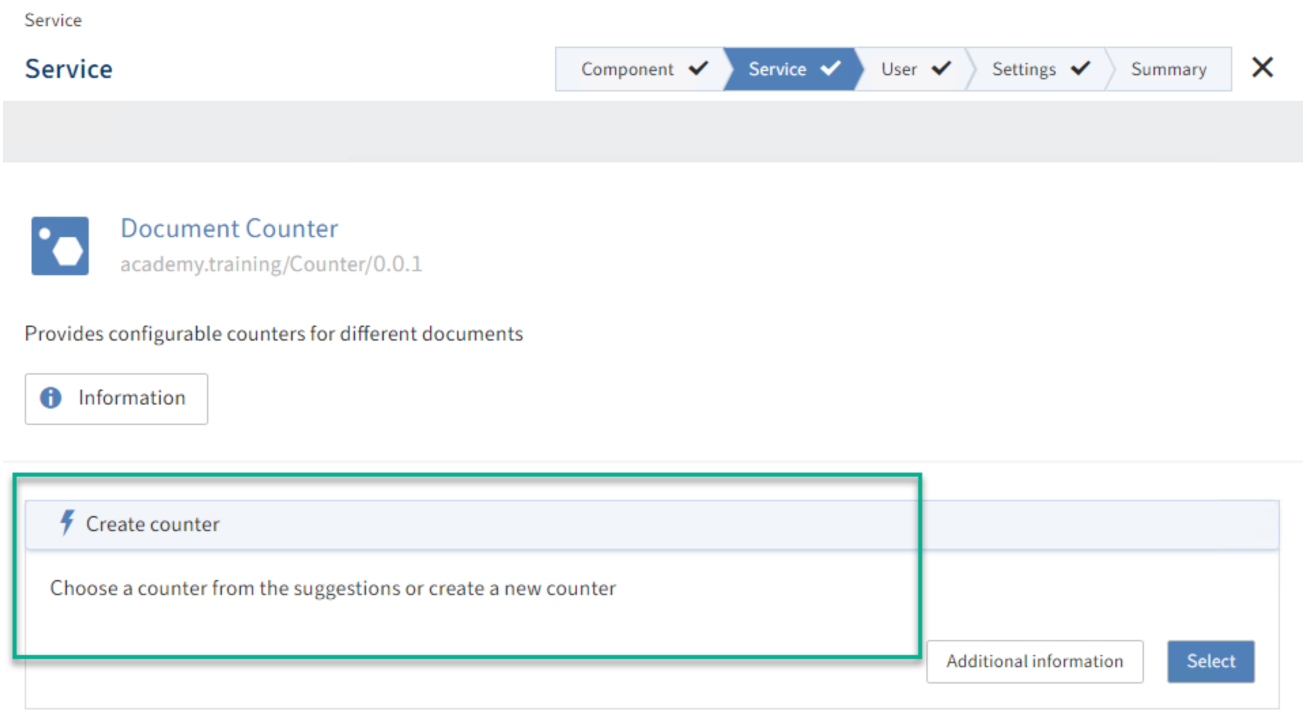
**Please note**

Triggers are always defined in the component class.

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName = "D
public class CounterComponent {

  @Trigger(displayName="Call document counter")
  @WebHook(endpoint="createCounter")
  public CounterTriggerData triggerCounter(@Config CounterTriggerData triggerData) {
    //Implementation
  }

  @Trigger(…)
  @Scheduled(…)
  public TriggerData otherTrigger(@Config TriggerData triggerData) {
  //Implementation
  }
}
```

## Service

Services provide different functions in ELO Flows. If a component is not used as the first component in ELO Flows, then it provides different services for selection. If multiple services are available, they can also be grouped.



Provides configurable counters for different documents

Create counter

Choose a counter from the suggestions or create a new counter

**Please note**

Services are always defined in the component class.

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName = "D
public class CounterComponent {
  @Service(…)
  public ServiceOutputData createCounter(SerivceInputData ) {
    //Implementation
  }
}
```

## Input fields

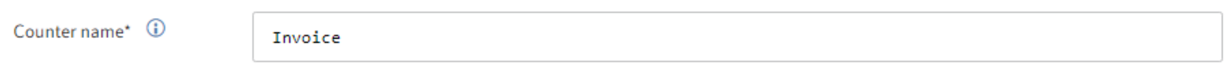
The input fields are used to capture user data. The fields can have different types and also offer suggestions for input. Data from previous components can also be used via keys via the input fields. The input fields are transferred to a service `@Service` or trigger`@Trigger`, where they are used for configuration tasks over the course of the flow.

The following types are possible:

- String
- Integer
- Boolean
- Object
- Array

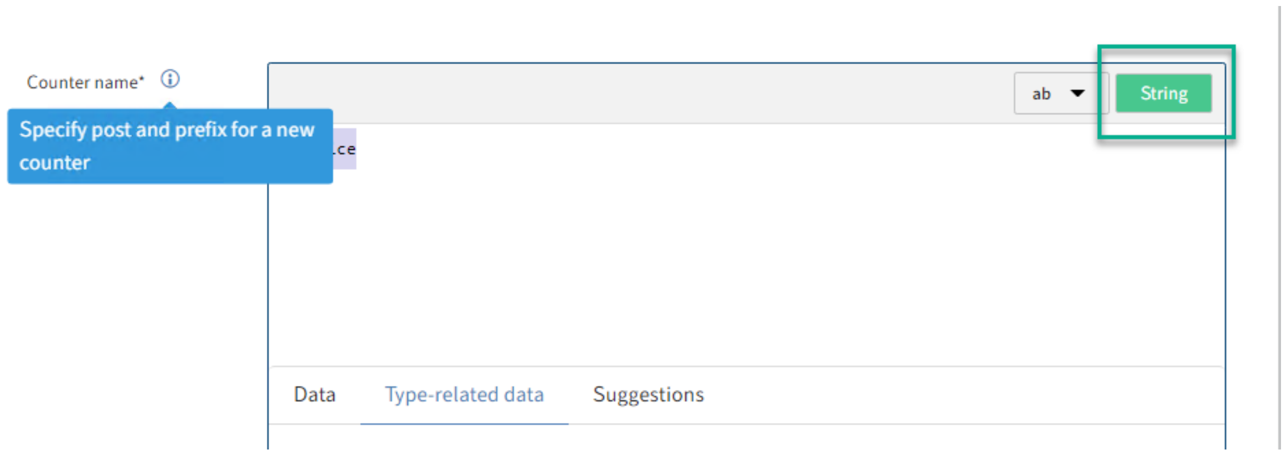
### String input field (text)

The input fields for values of different types also include the configuration tree and a JSONata editor.


Counter name*

Invoice

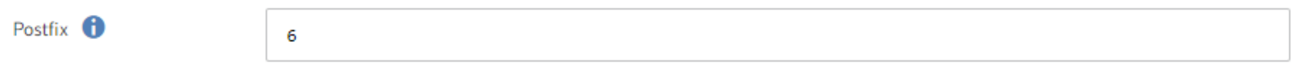
To implement an input field, a class is declared that provides the fields via the annotation `@Property`. This class is transferred to a service/trigger method as a parameter. The variable type (*String counterName* under annotation `@Property`) determines the type of the field in the editor.



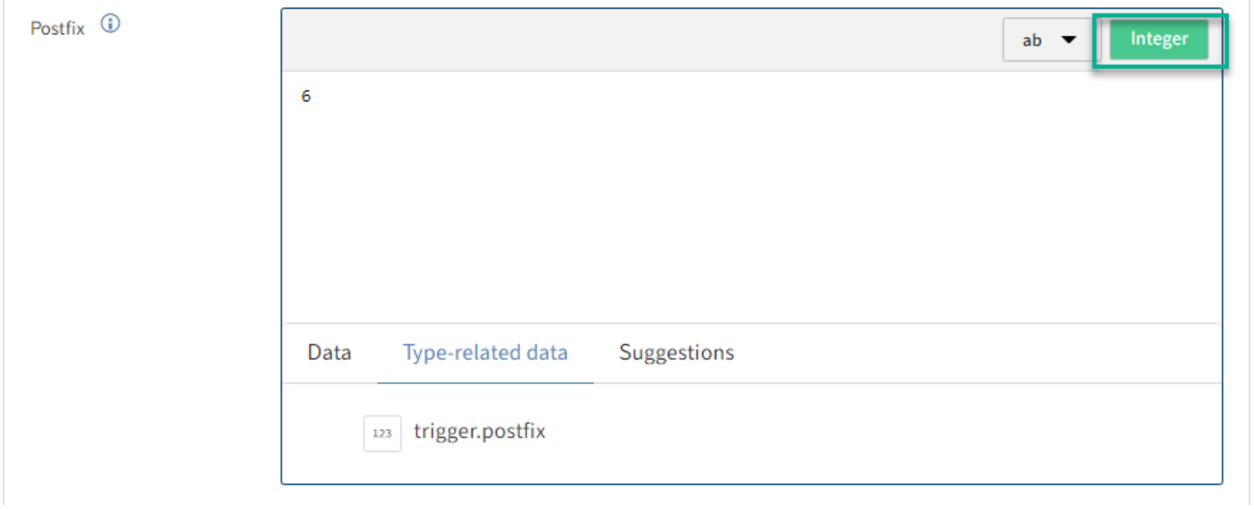
```
CounterInput.java
public class CounterInput {
  @Property(displayName="Counter name" description="Enter postfix and prefix for new counter")
  private String counterName; // Type string in input field editor
  //Implement getter and setter methods
}

CounterComponent.java
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName = "D
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
}
```

### Integer input field

To implement an integer type input field, proceed in the same way as for a string input field. The variable type, in our example *postfix*, is declared by the `int` type.

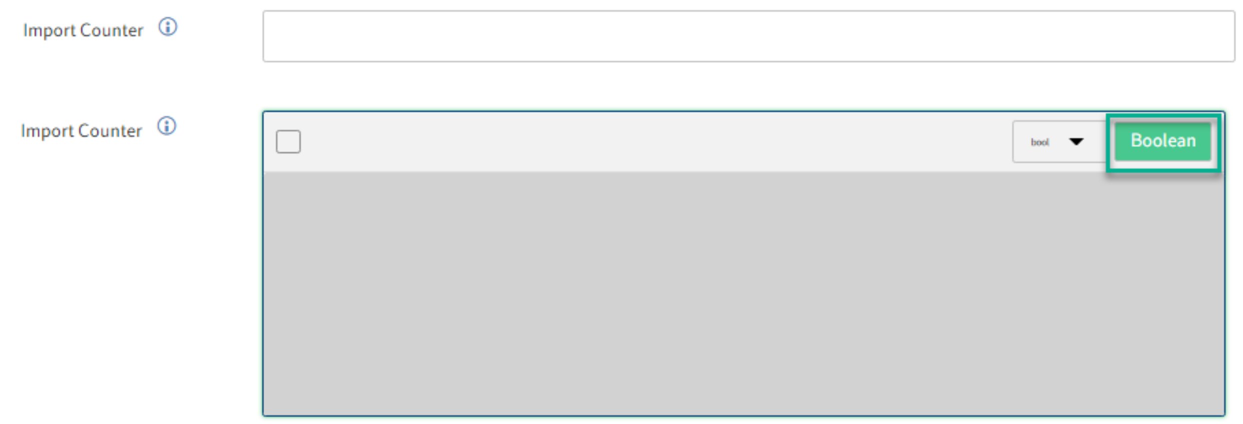

Postfix

6



```
CounterInput.java
public class CounterInput {
  @Property(displayName="Postfix")
  private int postfix; //Type Integer in the input field editor
  //Implement getter and setter methods
}

CounterComponent.java
@Component(version = "0.0.1", namespace = "academy.training",
name = "Counter", displayName = "Document counter",
description = "Unique allocation of identification numbers")
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
}
```

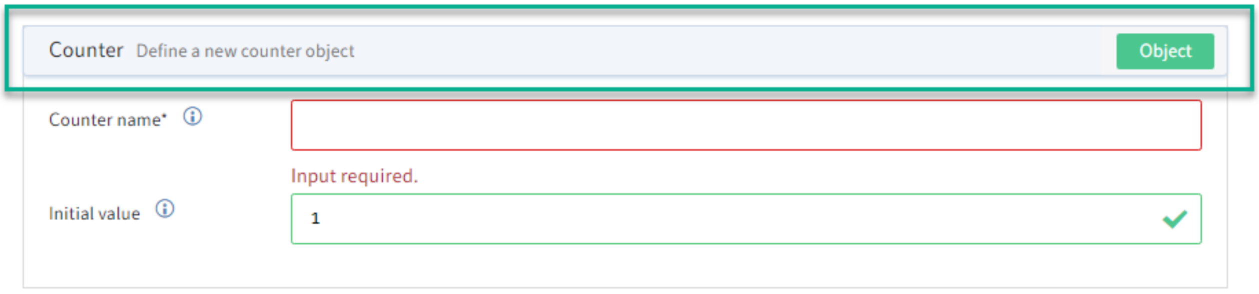
### Boolean input field

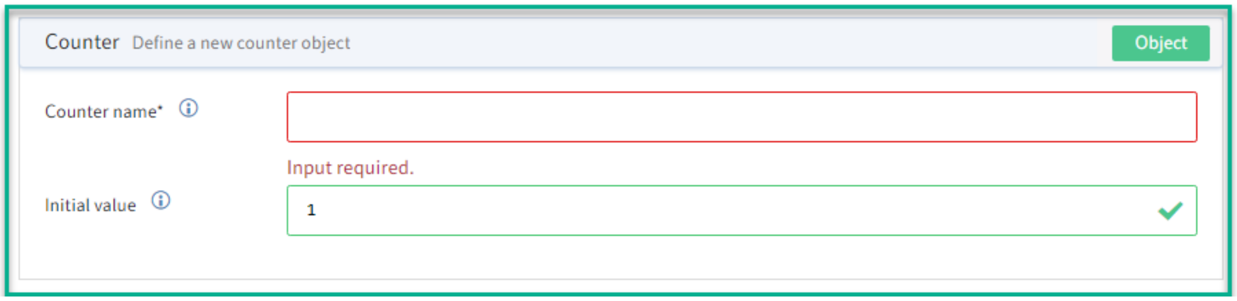


```
CounterInput.java
public class CounterInput {
  @Property(displayName="Import counter")
  private boolean importCounter; //Type Boolean in the input field editor
  //Implement getter and setter methods
}

CounterComponent.java
@Component(version = "0.0.1", namespace = "academy.training",
name = "Counter", displayName = "Document counter",
description = "Unique allocation of identification numbers")
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
  }
}
```

### Object input field

If you need multiple input lines at once (including of different types), you can implement an *Object* type input field. This allows you to combine multiple fields into an object.



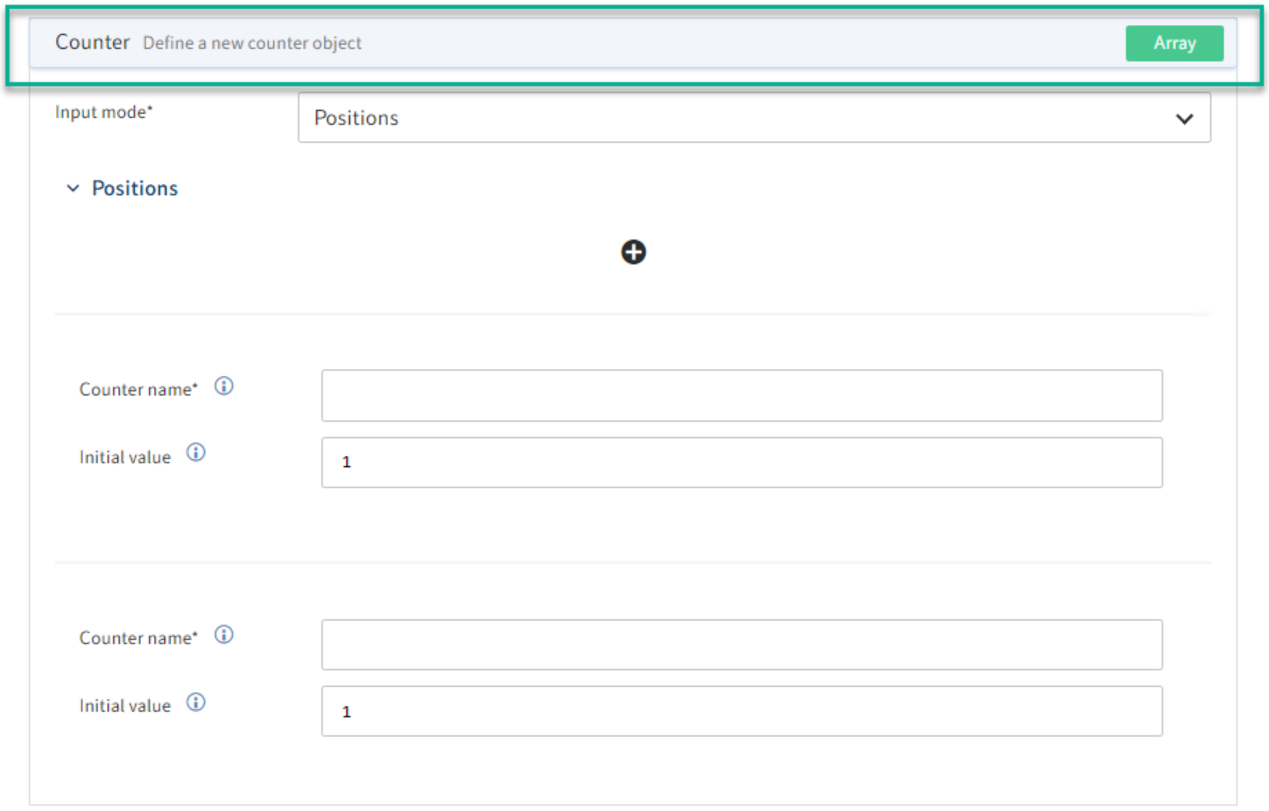


The example shows implementation of the *CounterObject* class as an internal class.

```
public class CounterInput {
  @Property(displayName="Define counter", description="Define a counter. Name and initial value"
  private CounterObject counterObj;//Type CounterObject in the input field editor
  class CounterObject {
    @Property(displayName="Initial value", description="The initial value is optional. The defau
    @DisplayOptions(order = 2, suggestValue = true)
    private int value = 1;

    @Property(displayName="Counter name", description="Enter the counter name"
    @DisplayOptions(order = 1)
    private String counterName;

    //Implement getter and setter methods
  }
}

@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName = "D
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
}
```

### Array input field

If multiple fields of one type (string, Boolean, integer, or object) are required, they can be combined into an array. In the example below, multiple object input fields are grouped.

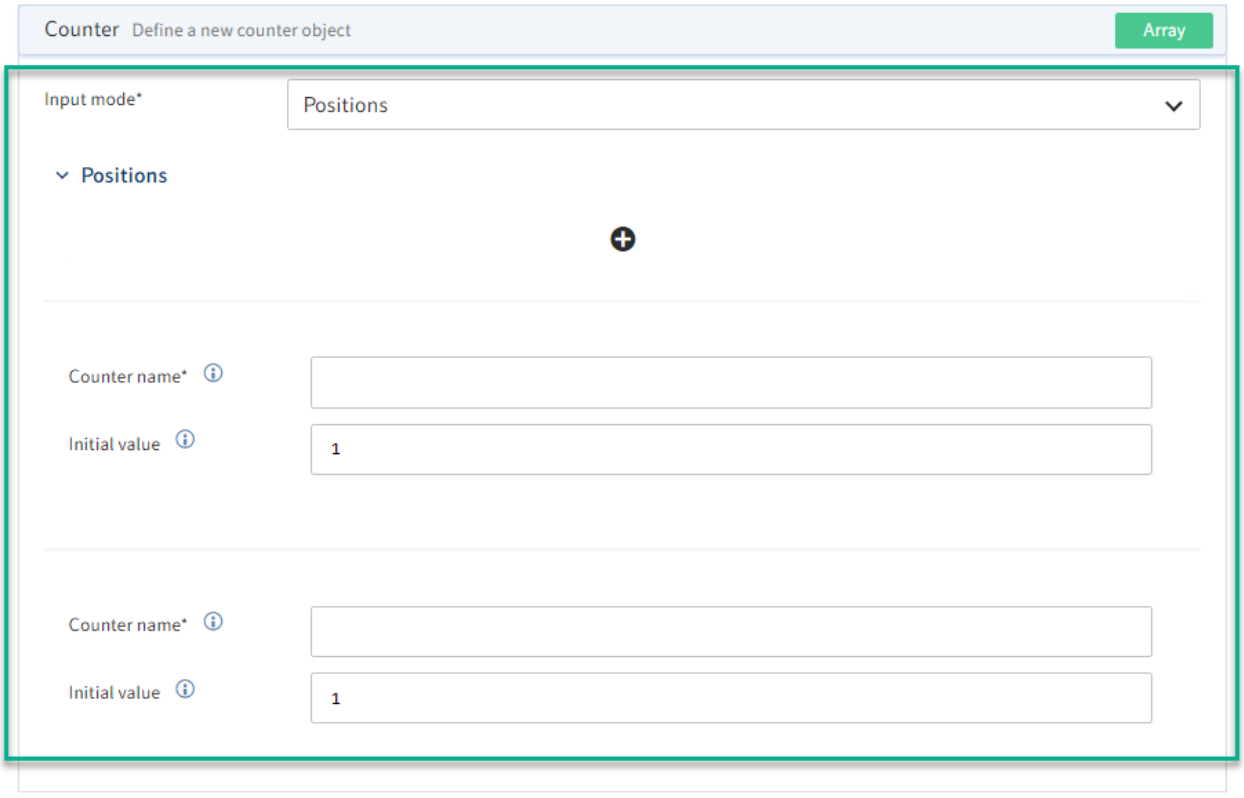

Counter Define a new counter object
Array

Input mode*: Positions

Positions

Counter name*:
Initial value: 1

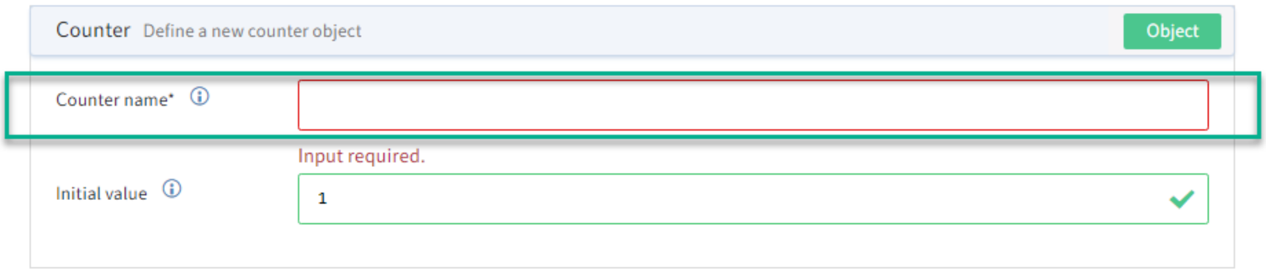
Counter name*:
Initial value: 1



```
public class CounterInput {
  @Property(displayName="Define counter", description="Define a counter with name and initial va
  private CounterObject[] counterObjects;//Type CounterObject[] in the input field editor

  class CounterObject {
    //See object input field
  }
}
```

### Mandatory fields

All input fields can be marked as mandatory. The field name is then marked with an asterisk (*). This prevents the user from leaving the input screen if these entries are missing.

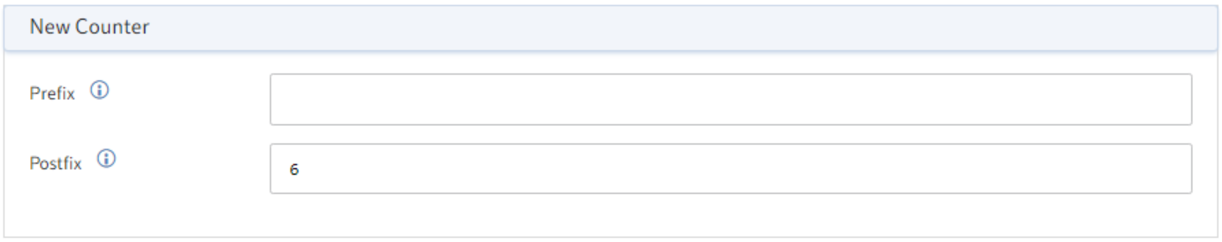


Setting the attribute `required` to `true` in the `Property` annotation marks the field as mandatory.

```
@Property(displayName="Counter name" description="Enter postfix and prefix
for new counter", required=true)
private String counterName; // Type string in input field editor
```

## Order, default field values

The fields can also be configured via the `@DisplayOptions` annotation. You can configure value suggestions (suggestValue) or the order (order) of the fields here.

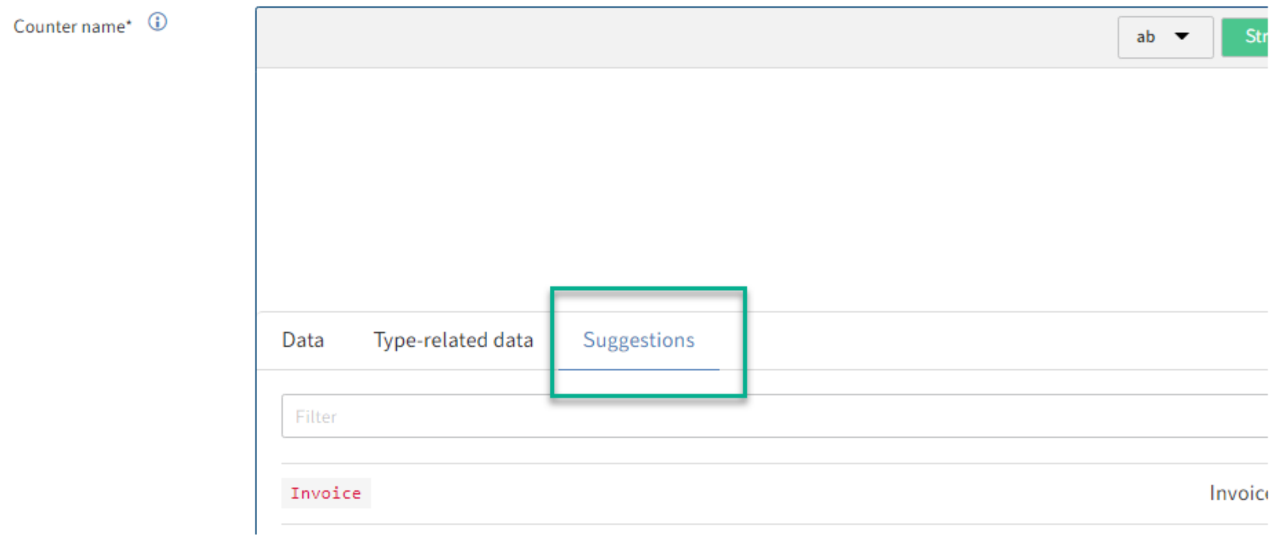

New Counter

| | |
|---|---|
| Prefix | |
| Postfix | 6 |

```
@Property(displayName="Postfix")
@DisplayOptions(order=2, suggestValue=true)
private int postfix = 6;

@Property(displayName="Prefix")
@DisplayOptions(order=1, size=2)
private String prefix = "Invoice";
```

## 'Suggestions' tab

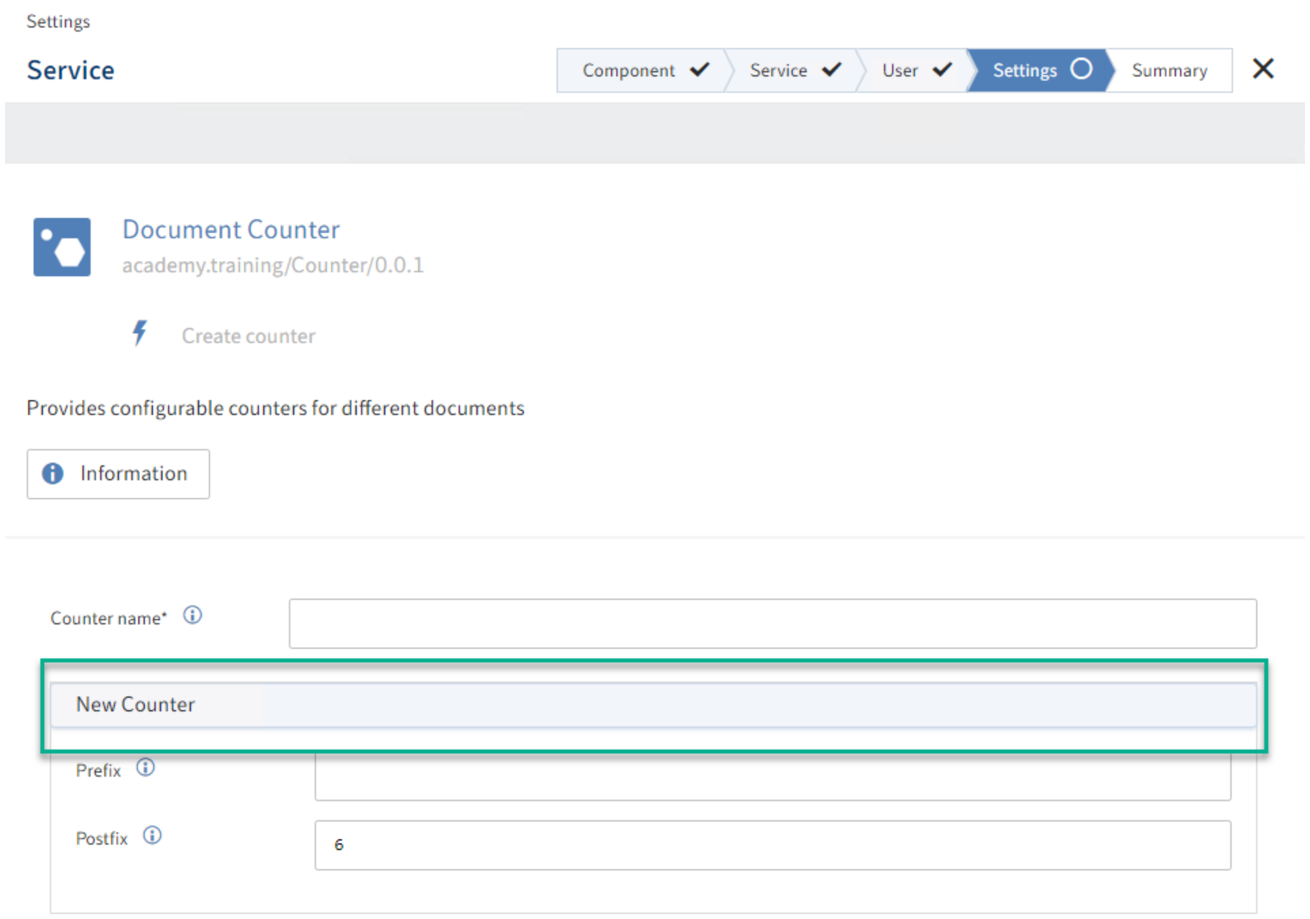
In all input fields, suggestion lists can be filled via the annotations `@Lookup` and `@LookupProvider`. The function with the annotation `@LookupProvider` must be implemented in the main component class `@Component`.



```java
public class CounterInput {
  @Property(displayName = "Counter name", required=true)
  @Lookup("getCounters")
  private String counterName;
}

@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName = "D
public class CounterComponent {

  @LookupProvider("getCounters")
  public Map<String,String> getCounters() throws CounterException {
  HashMap<String,String> map = new HashMap<>();
    //Implementation
    return map;
  }
}
```

## Field groups

The input fields can be combined into groups by topics. In the figure, the fields *Prefix* and *Postfix* are combined into the group *New counter*.



Document Counter
academy.training/Counter/0.0.1

Create counter

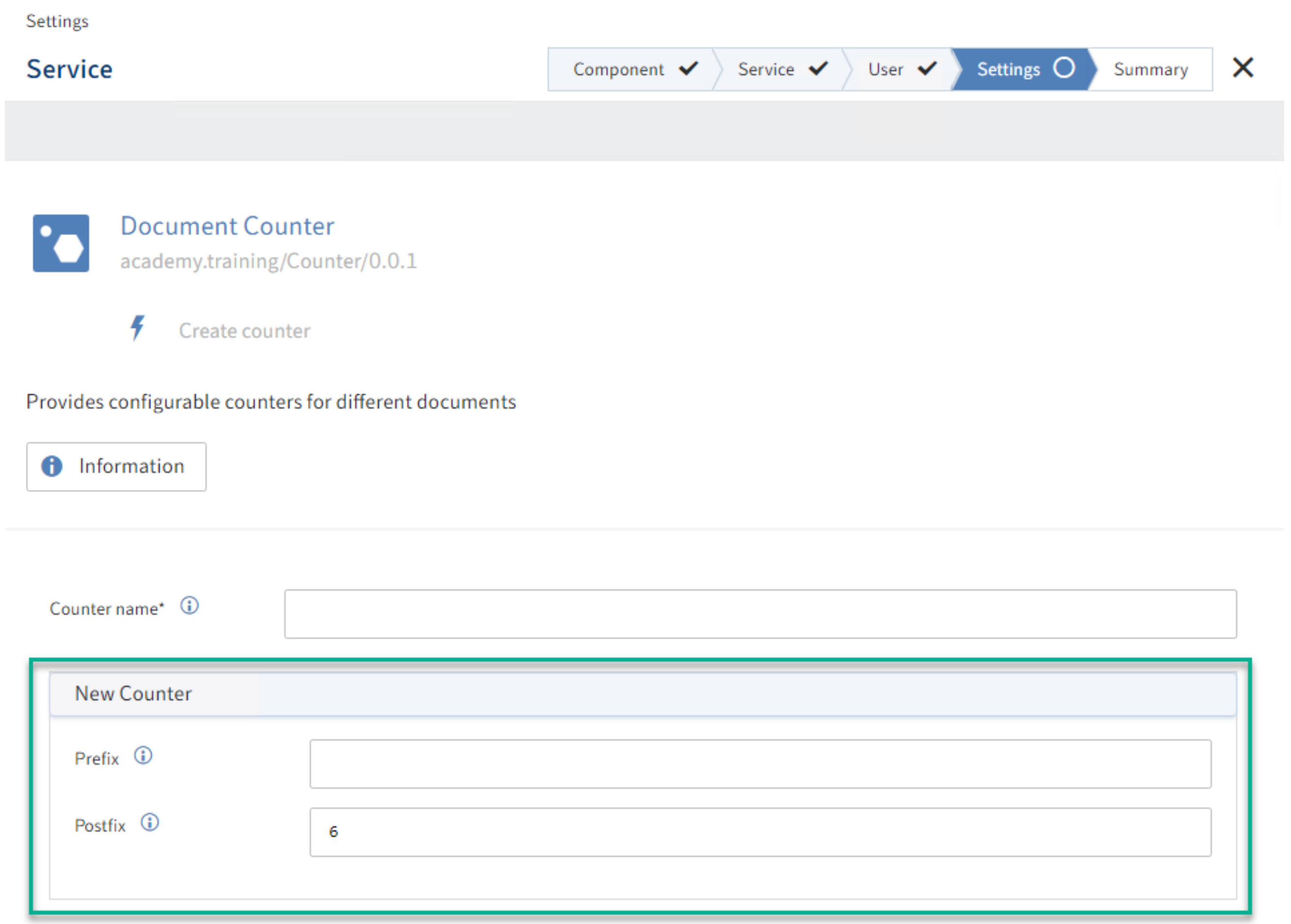
Provides configurable counters for different documents

Counter name*

New Counter

Prefix

Postfix 6



Provides configurable counters for different documents

Counter name*

New Counter

Prefix

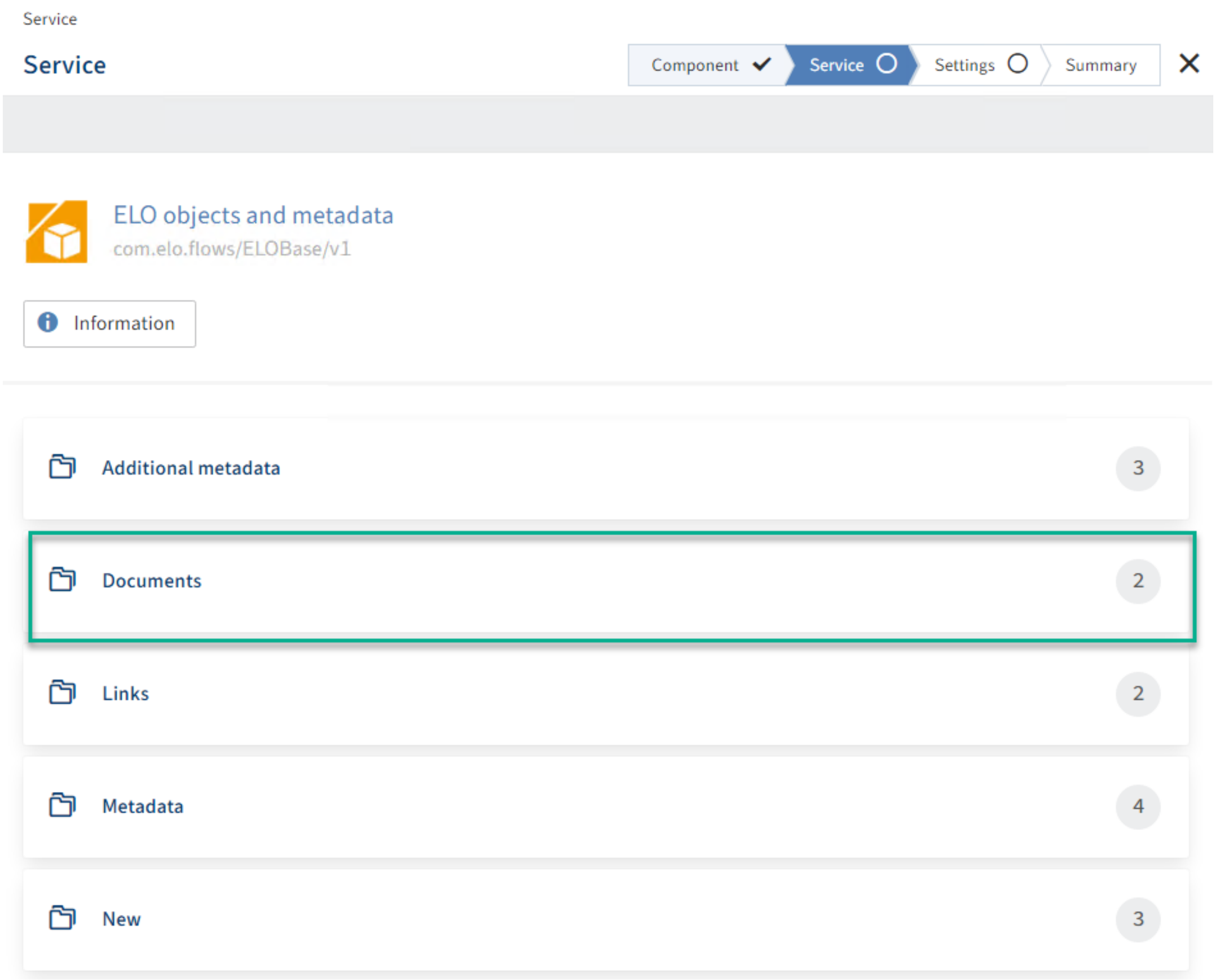
Postfix: 6

To group input fields, the annotations `@PropertyGroups` and `@PropertyGroup` are added to the class declaration. The annotation `@PropertyGroupRef` is also added to the fields that are combined into a group with the corresponding group name.

```
@PropertyGroups(@PropertyGroup(displayName="New counter", name="counterGroup"))
public class CounterInput {

  @Property(displayName = "Counter name", required = true, description = "Select a counter name"
  @Lookup("getCounters")
  private String counterName;

  @Property(displayName = "Prefix", description = "Only in combination with counter name")
  @PropertyGroupRef("countergroup")
  @DisplayOptions(order = 1)
  private String prefix;

  @Property(displayName = "Postfix", description = "Only in combination with counter name")
  @PropertyGroupRef("countergroup")
  @DisplayOptions(order = 2, suggestValue = true)
  private int postfix = 6;
}
```

## Service groups

Different services, see also the *ELO objects and metadata* component, can be combined into groups.



To group the services, the services are implemented in individual classes. The annotation `@ComponentServices` is added to these classes. For the groups to be shown, at least two groups have to be defined (i.e. two classes with `@ComponentServices` and contained `@Services`). The annotation `@ComponentServices` must always refer to the main component class (in our case `CounterComponent`) via the `component` attribute.

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName = "D
public class CounterComponent {
}
```

```
@ComponentServices(component = CounterComponent.class, name = "CounterServices", displayName = "
public class CounterServices {

  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}

  @Service(name="ChooseCounter")
  public CounterOutput chooseCounter(CounterInput input) {}
}

@ComponentServices(component = CounterComponent.class, name = "CounterServices", displayName = "
public class CounterServicesImportGroup {

  @Service(name="ImportCounter")
  public CounterImportOutput importCounter(CounterImportInput input) {}

  @Service(name="FindImportCounter")
  public CounterImportOutput findImportCounter(CounterImportInput input) {}
}
```

## Indexserver connection

The connection to the ELO Indexserver can be established automatically via the annotation `@Connection` and `@ConnectionRequired`. The connection to the ELOix is declared via the annotation `@Connection` in the *IXConnection* type variable and can be used in all methods via the annotation `@ConnectionRequired`.

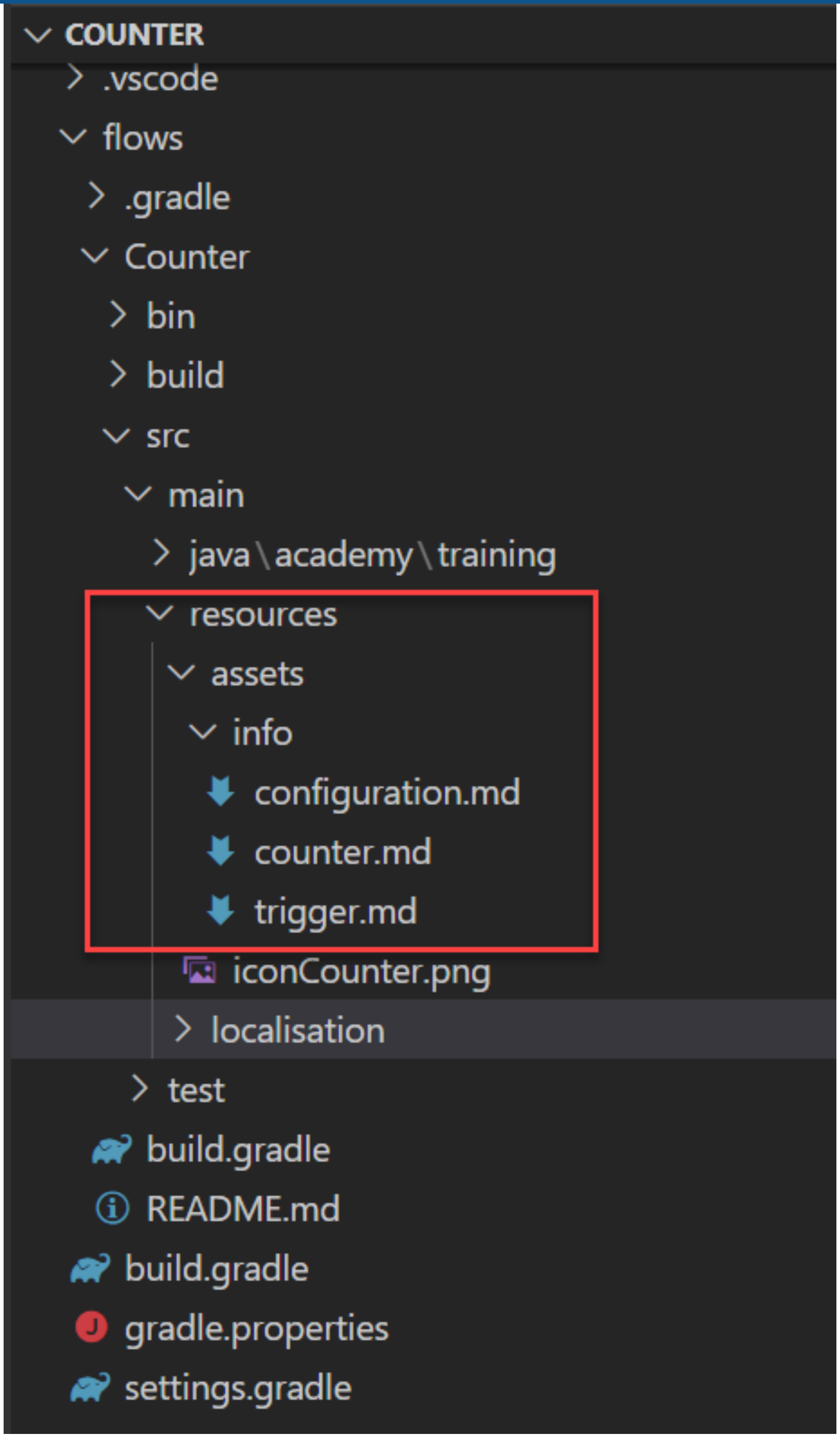
```
public class CounterComponent {

  @Connection
  IXConnection ixConnect;

  @LookupProvider("getCounters")
  @ConnectionRequired
  public Map<String,String> getCounters() {
    HashMap<String,String> map = new HashMap<>();
    //Implementation
    return map;
  }
}
```

## Component information

The documentation can be integrated in different positions, such as the service or component description, via the annotation `@Guide`. First, you will have to file the relevant markdown file in the corresponding project structure, which is then referenced later in the solution implementation.

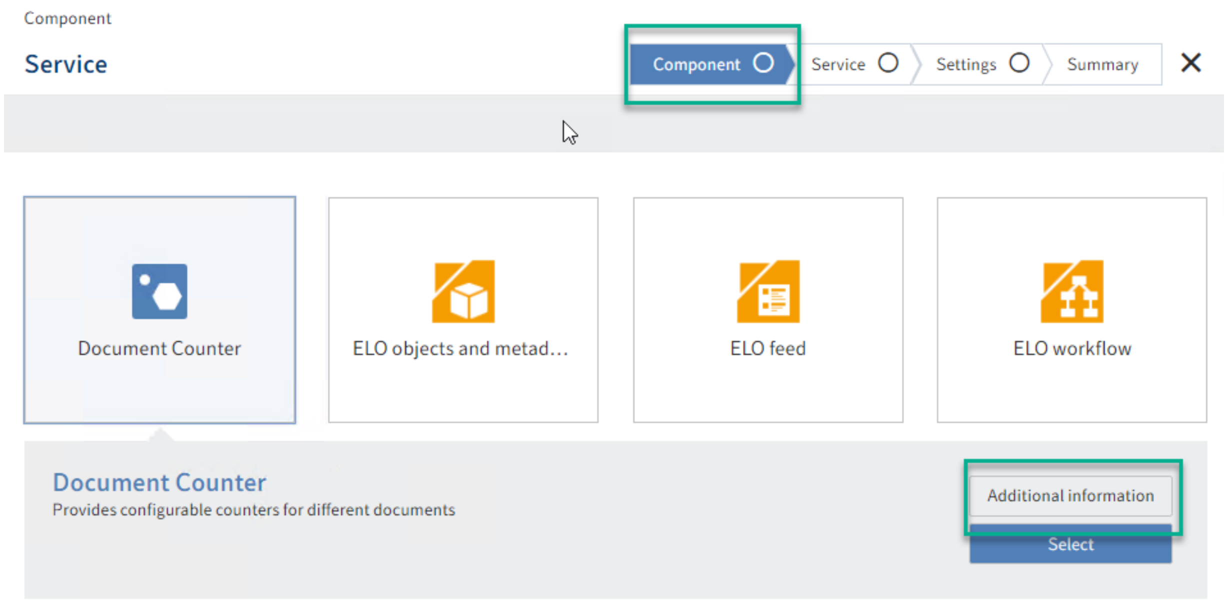
**Please note**

The markdown files are filed to the intended project structure (see figure). Otherwise, the documentation cannot be read or shown.



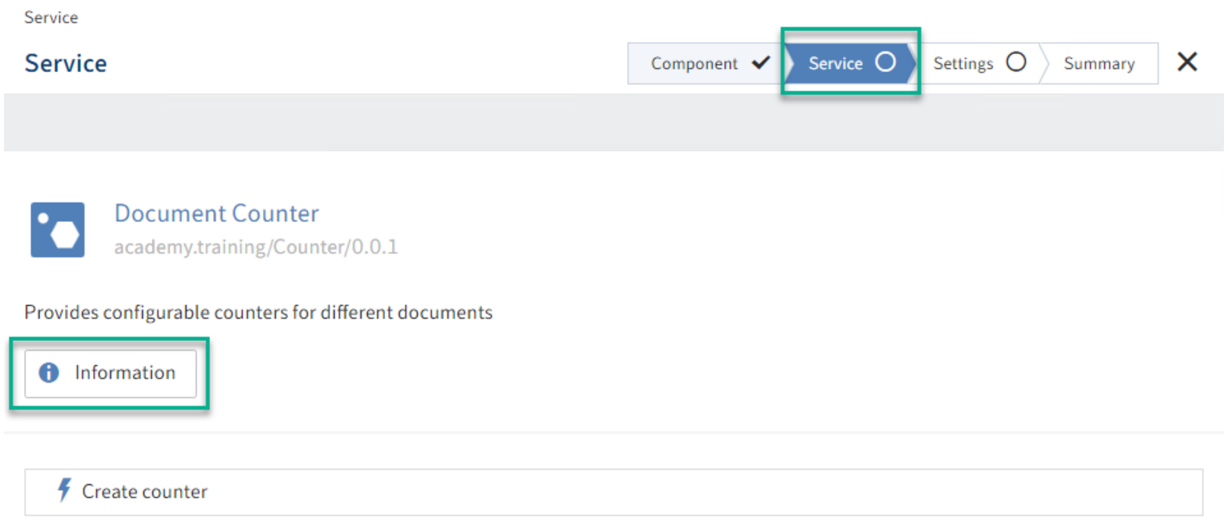
The following combinations with other annotations are possible:

1. Information on a component (`@Component` and `@Guide`). Visible in both the service and the trigger selection view.



Document Counter

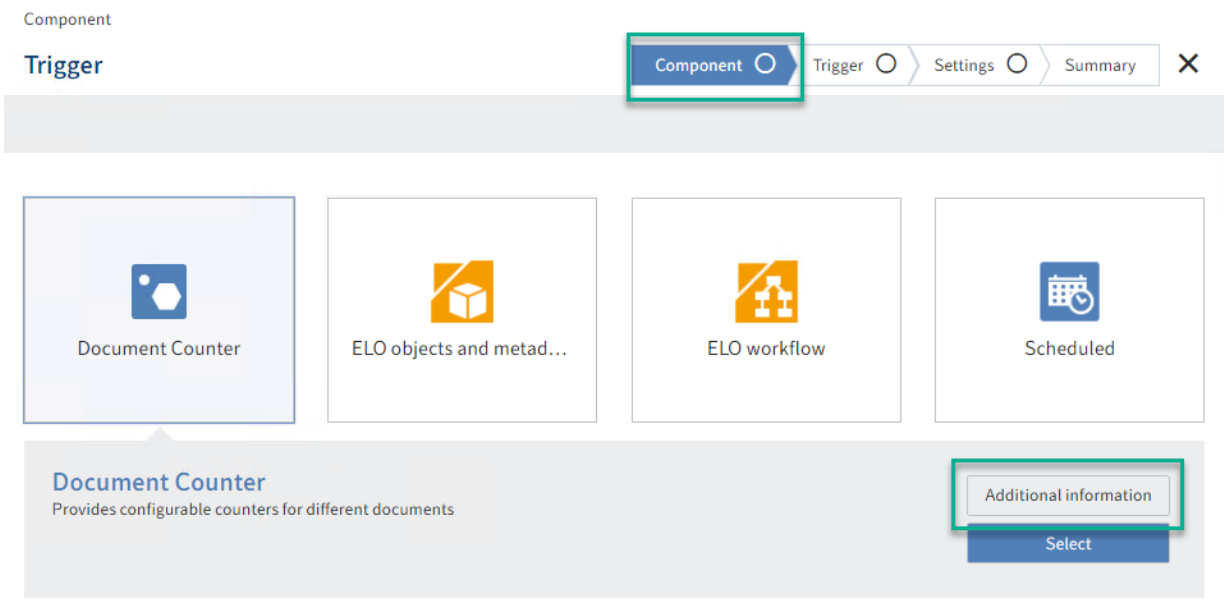
Provides configurable counters for different documents



Document Counter

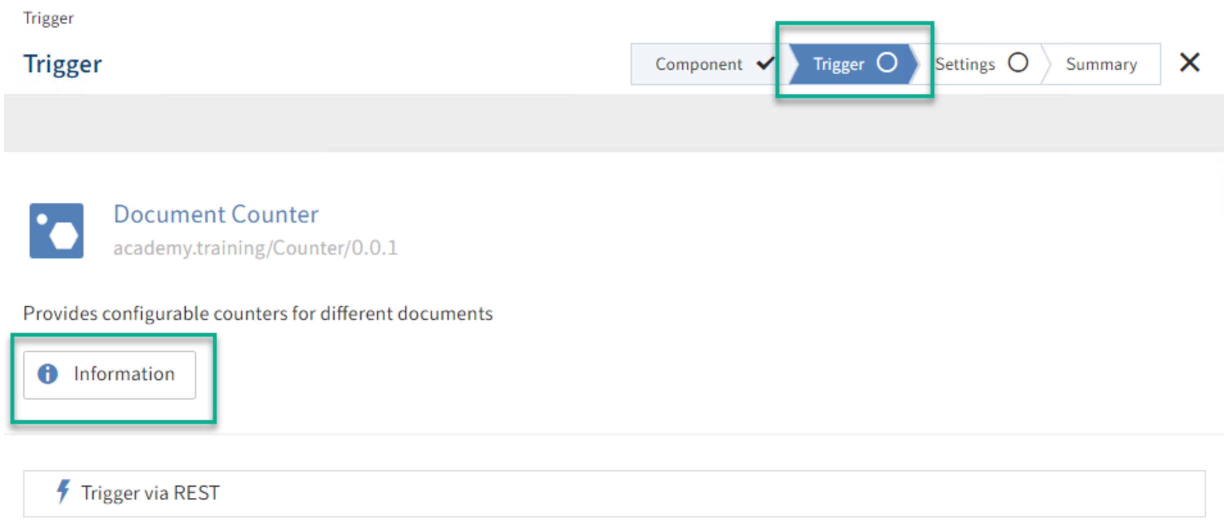
academy.training/Counter/0.0.1

Provides configurable counters for different documents



Document Counter

Provides configurable counters for different documents

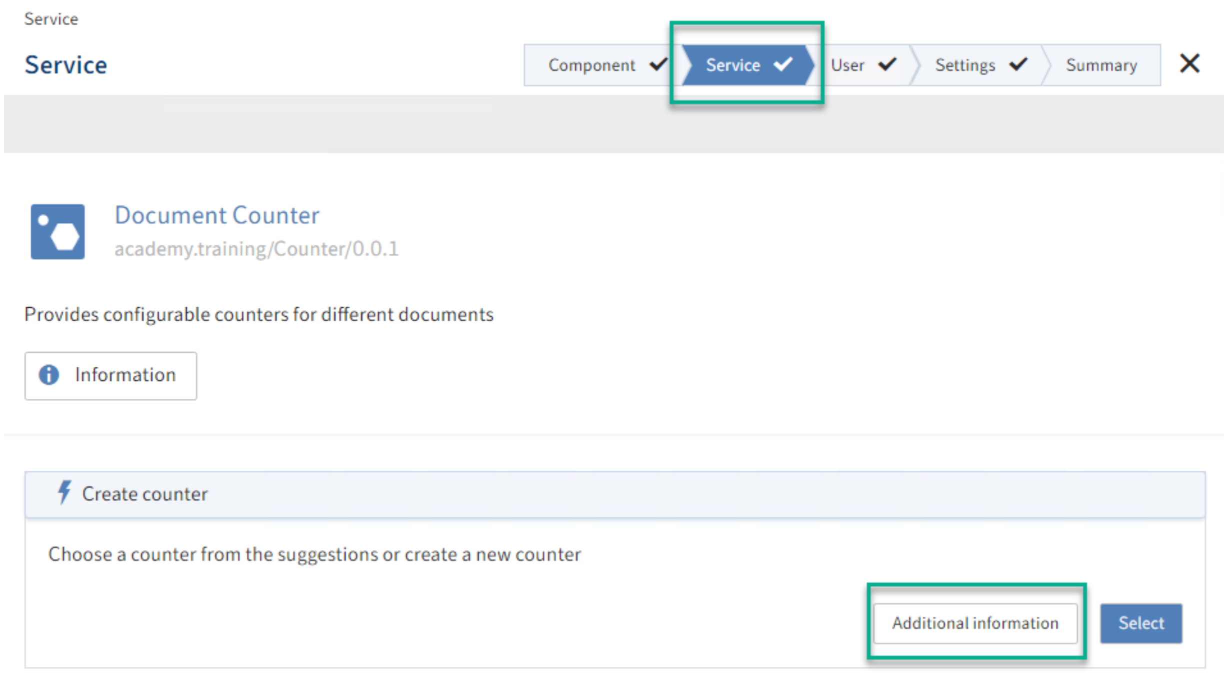


Document Counter

academy.training/Counter/0.0.1

Provides configurable counters for different documents

2. Information on a service (`@Service` and `@Guide`).



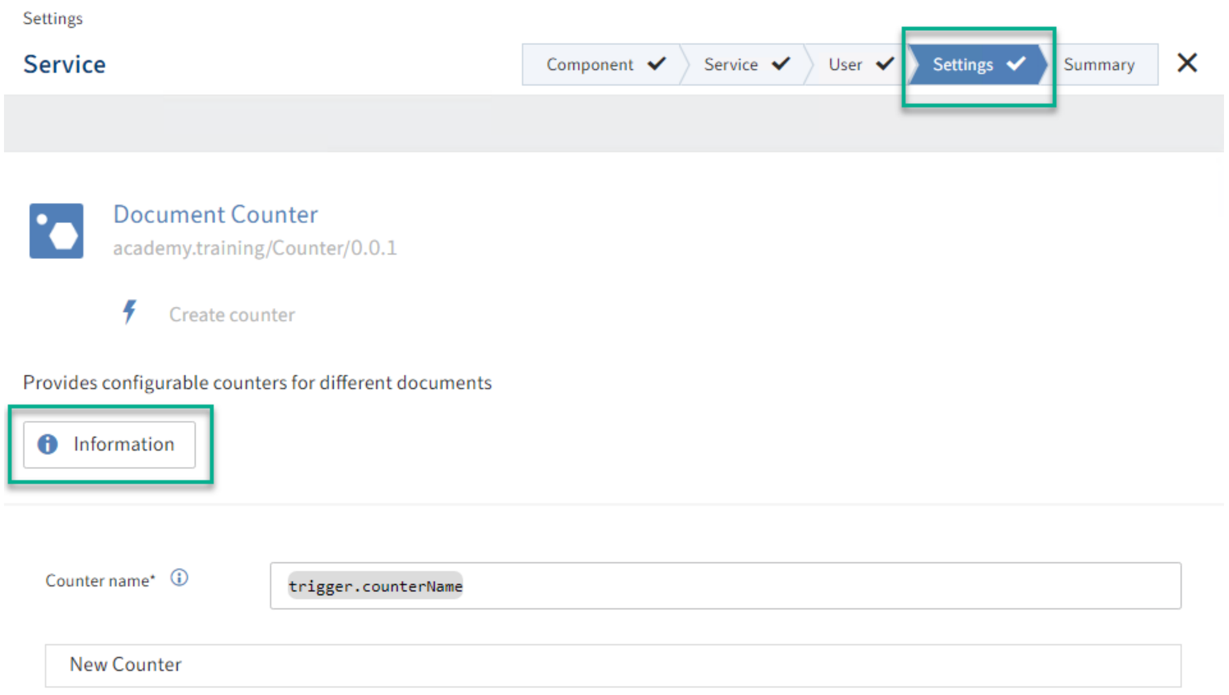
Document Counter

academy.training/Counter/0.0.1

Provides configurable counters for different documents

Create counter

Choose a counter from the suggestions or create a new counter



Document Counter

academy.training/Counter/0.0.1

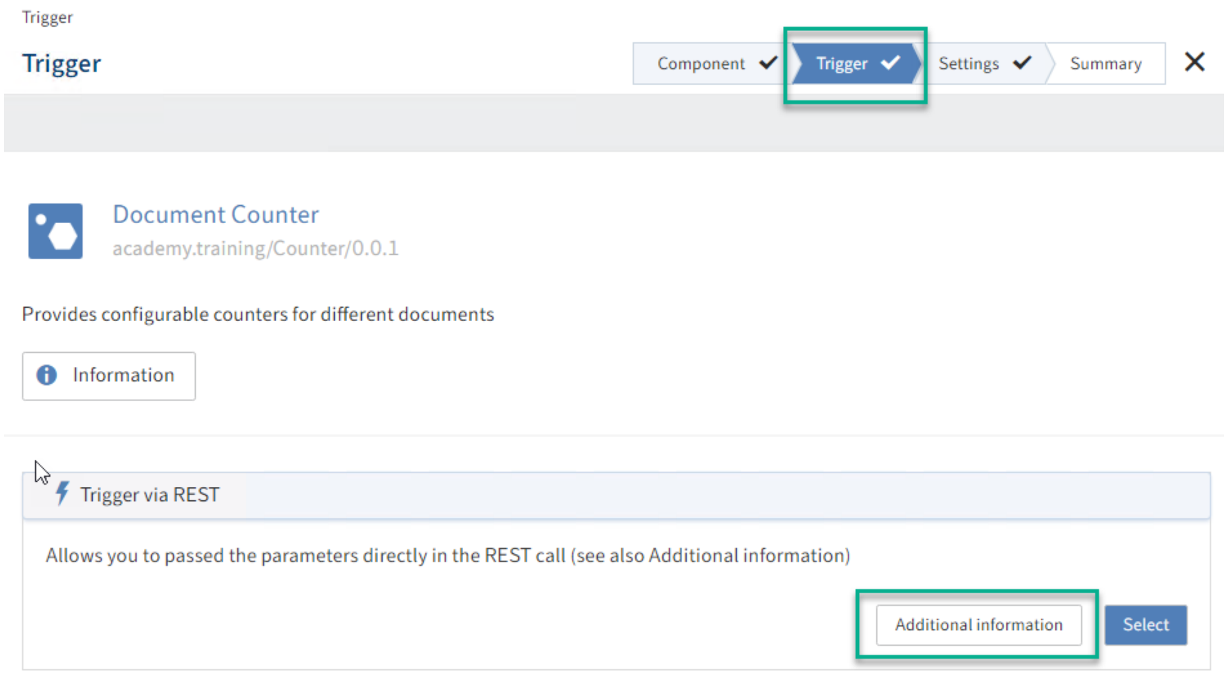
Create counter

Provides configurable counters for different documents

Counter name* `trigger.counterName`

New Counter

3. Information on a trigger (`@Trigger` and `@Guide`)



Trigger

Trigger

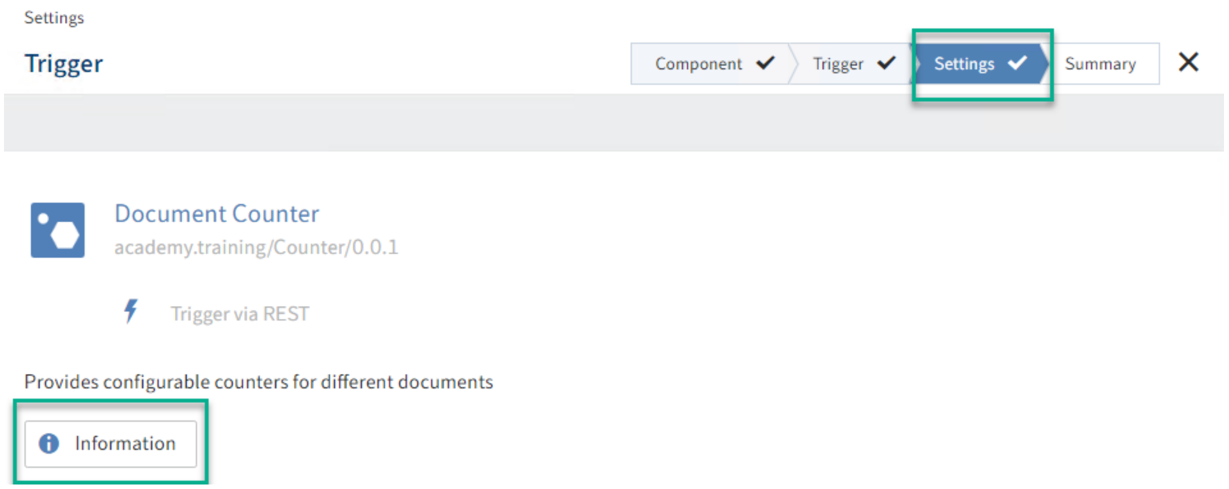
Document Counter

academy.training/Counter/0.0.1

Provides configurable counters for different documents

Trigger via REST

Allows you to passed the parameters directly in the REST call (see also Additional information)



Settings

Trigger

Document Counter

academy.training/Counter/0.0.1

Trigger via REST

Provides configurable counters for different documents

## Overview of annotations

An overview of the annotations available to us during development.

### @Component

```
package com.elo.flows.api.components.annotations;

public abstract @interface Component {
    public abstract java.lang.String namespace();
    public abstract java.lang.String name();
    public abstract java.lang.String version();
    public abstract java.lang.String displayName();
    public abstract java.lang.String iconUri() default "/assets/icon.png";
    public abstract java.lang.String description() default "";
}
```

| Name | Description |
|---|---|
| *namespace* | Namespace (package in project). |
| *name* | Component name. |
| *version* | Current version. |
| *displayName* | Display name that is shown in the ELO Flows administration area. |
| *iconUri* | Path to the component icon, default /assets/icon.png. |
| *description* | Brief description of the component shown when the component is selected. More information is entered via the annotation `@Guide` in the component. |

### @Service

The service provided in a component.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Service {
    public abstract java.lang.String name() default "";
    public abstract java.lang.String displayName() default "";
    public abstract java.lang.String description() default "";
}
```

| Name | Description |
|---|---|
| *name* | Name of service. |
| *displayName* | Display name that is shown in the ELO Flows administration area. |

| Name | Description |
|---|---|
| *description* | Short name of the service shown when the service is selected. More information is entered via the annotation `@Guide` in the service. |

## @Trigger, @WebHook

Trigger that can be used to start a flow.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Trigger {
    public abstract java.lang.String name() default "";
    public abstract java.lang.String displayName() default "";
    public abstract java.lang.String description() default "";
}

package com.elo.flows.api.components.annotations;

    public abstract @interface WebHook {
    public abstract java.lang.String endpoint() default "";
}
```

| Name | Description @Trigger |
|---|---|
| *name* | Name of the trigger. |
| *displayName* | Display name that is shown in the ELO Flows administration area. |
| *description* | Short name of the trigger shown when the trigger is selected. More information is entered via the annotation `@Guide` in the trigger. |

| Name | Description @WebHook |
|---|---|
| *endpoint* | Identifier for the call. |

## @Config

Used in a trigger (trigger method) as a parameter. Indicates that configurations can be made in the trigger on the *Settings* tab.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Config {
}
```

## @Synchron

Indicates a synchronous trigger.

```
package com.elo.flows.api.components.annotations;
public abstract @interface Synchron {

  public abstract  java.lang.Class<? extends java.lang.Object>[] resultClasses() default {};
}
```

## @Guide

Enables integration of information files in markdown format.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Guide {
    public abstract java.lang.String value();
}
```

## @Property

Field on the *Settings* tab.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface Property {
    public abstract java.lang.String displayName() default "";
    public abstract java.lang.String description() default "";
    public abstract boolean required() default false;
}
```

| Name | Description |
|---|---|
| *displayName* | Display name that is shown in the ELO Flows administration area. |
| *description* | Brief description of the input field shown next to the (i) as a tooltip. |
| *required* | Indicates whether the field is mandatory. |

## @DisplayOptions

Additional configuration options for input fields.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface DisplayOptions {
    public abstract int size() default (int) 1;
    public abstract int order() default (int) 2147483647;
    public abstract boolean hidden() default false;
    public abstract boolean suggestValue() default false;
}
```

| Name | Description |
|---|---|
| *size* | Size of the input field. |
| *order* | In case of multiple input fields in a group, you can choose an order for display. |
| *hidden* | A field can be marked as hidden. |
| *suggestValue* | The value of the @Property variables is automatically applied to the input field. |

## @Lookup

Suggestions tab in the input field. Used with the `@Property` annotation. You also have to make sure that a method is implemented via the annotation `@LookupProvider` that provides the content for the suggestions.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface Lookup {
    public abstract java.lang.String value();
}
```

## @LookupProvider

Suggestions in the input field. Used in combination with the `@Lookup` annotation.

```
package com.elo.flows.api.components.annotations;

public abstract @interface LookupProvider {
    public abstract java.lang.String value();
}
```

## @Connection

ELO Indexserver connection.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Connection {
    public abstract java.lang.Class<?> provider() default java.lang.Void;
}
```

## @ConnectionRequired

Used with `@Connection`, for example if the connection to the ELO Indexserver is used in a method.

```
package com.elo.flows.api.components.annotations;
    public abstract @interface ConnectionRequired {
}
```

## @PropertyGroups, PropertyGroup, PropertyGroupRef

The annotations are used to combine the different services into groups on the *Services* tab.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface PropertyGroups {
    public abstract com.elo.flows.api.schema.annotations.PropertyGroup[] value();
}

package com.elo.flows.api.schema.annotations;

public abstract @interface PropertyGroup {
    public abstract java.lang.String displayName();
    public abstract java.lang.String name();
    public abstract boolean collapsed() default false;
    public abstract java.lang.String description() default "";
    public abstract int order() default (int) 2147483647;
}
```

| **Name** | **Description** |
|---|---|
| *displayName* | Display name that is shown in the ELO Flows administration area. |
| *name* | Group name. |
| *collapsed* | This group can be opened when making a selection. |
| *description* | Brief description of the group. Shown next to the group name when the group is open. |
| *order* | If there are multiple settings in the group, you can determine the order. |

```
package com.elo.flows.api.schema.annotations;

public abstract @interface PropertyGroupRef {
    public abstract java.lang.String value();
}
```