

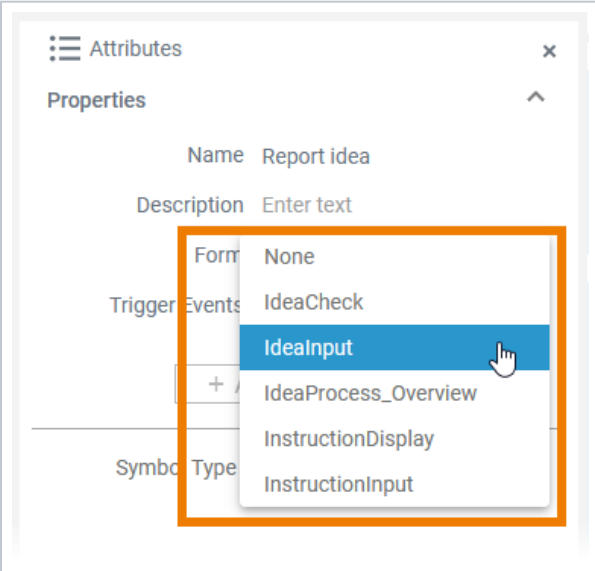
Using Forms

Processes are based on data that is going in, is processed, and coming out. **Scheer PAS Designer** allows to create forms that can be used to provide data into a process, or to display data from a process in the generated application.

How to design a form is described in detail on [Modeling Forms](#).

Assigning a Form to a User Task

In a BPMN process, forms can be assigned to [user tasks](#). When the related user task is reached during process execution, the application will show the form and wait for user input.



Select the user task and switch to the **Attributes** panel. Then, select the form you want to use with this user task from the **Form** selection list.

Select **None** if you want to remove the form at all from the user task.

Form Elements in the Data Model

In most cases, when designing a form (see [Modeling Forms](#)), you will also be using form elements that can hold data:

- [button](#) (pushed true/false)
- [checkbox](#) (true/false)
- [date picker](#) (can hold a timestamp)
- [file upload](#) (can hold a list of links to files in the file storage)
- [radio button group](#) (can hold a selected value)
- [select field](#) (can hold a selected value)
- [text box](#) (can hold an arbitrary text)
- [text area](#) (can hold an arbitrary text)

These form elements are to be accessible from the execution diagram to read the contents or to be prepopulated.

Each form generates a form class <name of the form> to a dedicated package **Forms** in the **Data Model**. All above mentioned form elements - if used on the form - are generated as properties to the form class.

On this Page:

- [Assigning a Form to a User Task](#)
- [Form Elements in the Data Model](#)
- [Handling Form Data](#)
 - [Prepopulate Data Into a Form](#)
 - [Process Data From a Form](#)
 - [Mapping Diagram](#)
 - [Action Script](#)

BPMN_User_Task_Example



Click the icon to download a simple example project that shows what you can do with **User Tasks** in **Scheer PAS Designer**.

Idea_Management_Example



Click the icon to download a simple example model that shows what you can do with **Lanes** and **Forms** in **Scheer PAS Designer**. It also contains a configured **instance list** and shows the usage of **trigger events**.

Related Pages:

- [Mapping Data Structures](#)
- [Setting Form Elements Dynamically](#)
- [PAS Designer User Guide](#)
 - [Modeling Forms](#)
 - [Configuring Form Elements](#)
 - [Modeling Execution](#)
 - [Modeling Data Mapping](#)

Idea_Management_Example

Search Criteria

+

Base Types

Connectors

+

Process

+

Forms

API

-

Implementation

-

Forms

+

IdeaCheck

-

IdeaInput

button_ideaSubmit: Boolean

employeeName: String

ideaDescription: String

personnelNumber: String

IdeaProcess_Overview

+

InstructionDisplay

+

InstructionInput

+

IdeaManagement

Libraries

In the example on the left, form **IdeaInput** contains a button (**button_ideaSubmit**) and three text boxes. You cannot change these classes.

Expert Advice

To better distinct buttons from other form elements in the **Data Model**, we recommend to put **button_** at the beginning of the element name.

Handling Form Data

User tasks can have two [execution diagrams](#):

- **Get Data** is executed before showing the form. You can use this execution to prepopulate form elements before showing the form. The **Get Data** execution has a generated return parameter of type form class (see [Form Elements in the Data Model](#) above).
- **On Exit** is executed after having shown the form. You can use this execution to persist or process data that has been entered to the form. The **On Exit** execution has a generated input parameter **message** of type form class (see [Form Elements in the Data Model](#) above).

Prepopulate Data Into a Form

You can set a default value in the **Attributes** panel of a form element. This default will be applied before displaying the form. Refer to [Configuring Form Elements](#) for more information on that.


This static default can be overwritten dynamically with data from the process using execution **Get Data**. This is possible for the following form elements:

Element	Type	Description
checkbox	Boolean	Provide whether the checkbox is checked or not.
date picker	DateT ime	Provide a timestamp.
radio button	String	Provide the selected option as a string.
select field	String	Provide the selected option as a string.
text box	String	Provide a text.

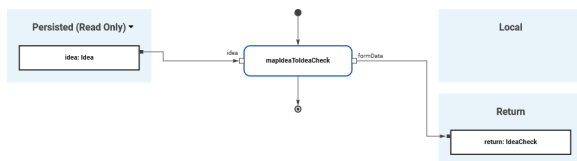
text area	String	Provide a text.
-----------	--------	-----------------


Expert Advice
 To get multi-line texts, you can use the [concat\(\) operation](#) in Action script to provide the newline symbol `\n`.


```
concat("first line", text("\n"), "second line");
```


Expert Advice
 You can not only set the contents of a form field but also override the specification of the form element itself like available options, alignment and more. How to do this is described on [Setting Form Elements Dynamically](#).

The data you want to push into the form must be available in the execution diagram, e.g. via a persisted variable, or via an operation.



The example above shows a **Get Data** execution from the [IdeaManagement](#) example. Form [IdeaCheck](#) is populated from a persisted variable `idea` that contains the necessary details. `mapIdeaToIdeaCheck` is a mapping operation that maps the persisted data to the form. Refer to [Modeling Data Mapping](#) for more information on how to create a mapping operation.

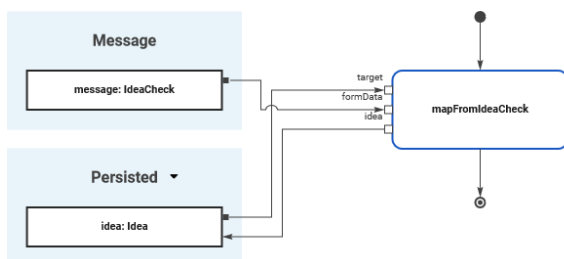


In this case, all relevant idea properties are directly mapped to the corresponding form elements. [Mapping Data Structures](#) explains how to perform more complex mappings if necessary.

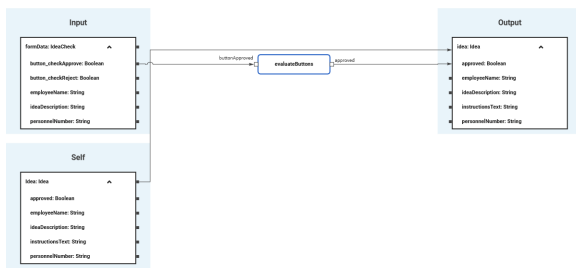
Process Data From a Form

Once the form has been submitted, execution **On Exit** is performed. Here, you can get data from the form and store or process it. To map the data, you can use a mapping diagram or action script.

Mapping Diagram



The example above shows an **On Exit** execution from the [IdeaManagement](#) example. Operation `mapFromIdeaCheck` implements a mapping diagram that maps the form data to the persisted idea.



Note the **Self** object on this operation. This operation must not be static but you need to implement it on the class you want to store the data to. The mapping will overwrite all values of the target class: All properties of the output class that are not mapped will be NULL after the mapping. So, the values that have been already stored need to be provided via the self object to the target class.

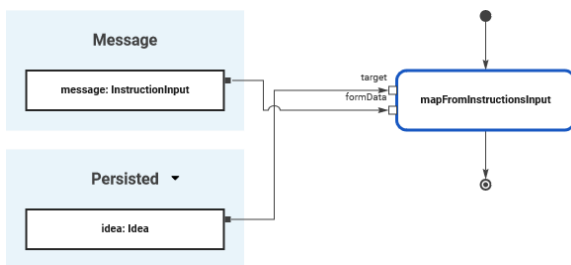


Note the order of execution for mapping diagrams:

1. Create statements.
2. Single field mappings.

Action Script

Alternatively, you can use [action script](#) to map the form data. This is implemented to operation [mapFromInstructionsInput](#) in the example model. [mapFromInstructionsInput](#) is a non-static operation on class [Idea](#)



It gets the self object and the form data as input, and maps the input data with the following action script:

```
set self.instructionsText = formData.instructionsText;
```