

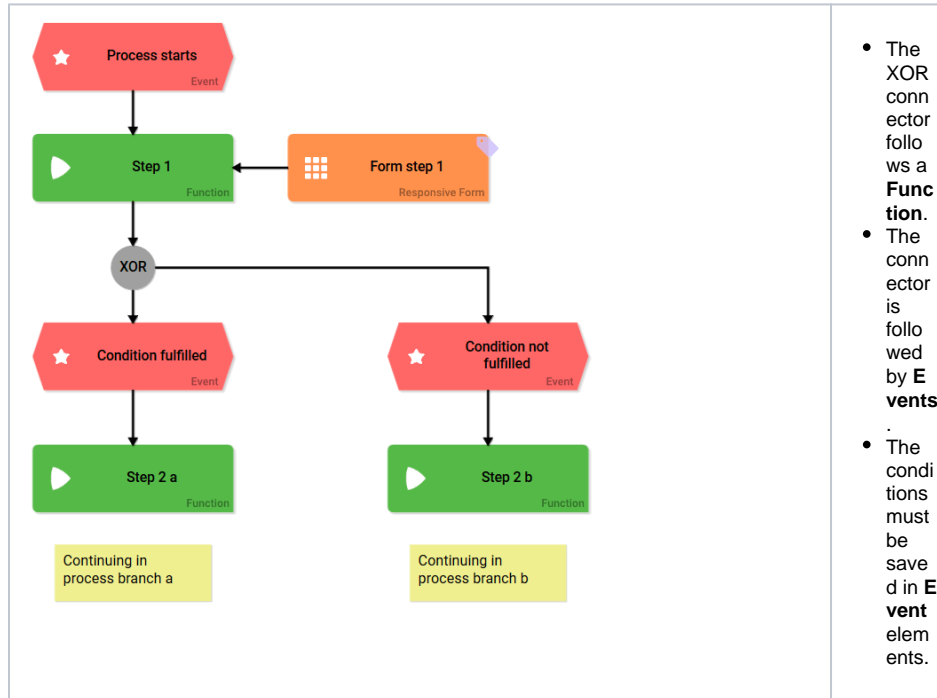
XOR Branching

XOR branching is always tied to **conditions**. These conditions are defined in the **Event** element and checked during process execution. The result determines which of the predetermined process branches is chosen. The XOR branching recognizes two results per condition: **Either** the defined condition is fulfilled **or** it is not fulfilled.



If the conditions were not defined correctly, then the process is stopped and the announcement **Error during model execution!** is displayed.

Conditions are defined in the element Event, therefore pay attention to the following during modeling:



In order to prevent an EPC from being ineffective always verify **event** and **counterevent** when designating constraints.

Example: A form offers the answers **YES** and **NO**. In the process, yes-answers shall follow a different path than no-answers. Therefore it has to be verified which of both paths needs to be followed after the form has been saved. Instead of checking if **YES or NO** was specified, check if **YES or NOT YES** was entered. This enables you to also pick up cases where neither answer was specified.

The counterevent to YES is not NO, but NOT YES!

On this Page:

- [Defining Conditions](#)
 - [Conditions: Syntax](#)
 - [Conditions: Special Cases](#)
- [Logical Operators](#)

Related Pages:

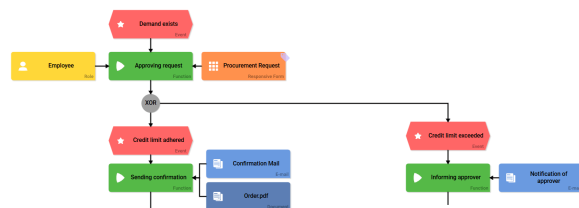
- [AND Branching](#)
- [EPC Elements](#)
 - [Connectors](#)
 - [Event](#)
- [Modeling Conventions](#)
- [Executing Processes](#)
 - [Possible Values: Value and Label](#)
- [Service](#)
 - [Procurement Process](#)

Related Documentation:

- [BPaaS Entwicklerhandbuch \(German\)](#)
 - [Procurement Process](#)

Defining Conditions

The **ACME** procurement process requires employees to complete a **Procurement Request** in order to react to a demand.



According to the guidelines of ACME Corp., the request can be immediately converted into an order if the order value does not exceed 50 Dollar.

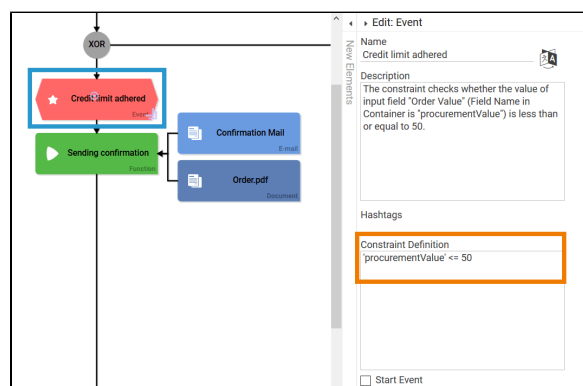
If the order contains items valued above 50 Dollar total, then the demand has to be verified prior to ordering. The employee enters the desired items into a Procurement Request, then an Order Value is determined.

The amount in **Order Value** determines the next process steps:

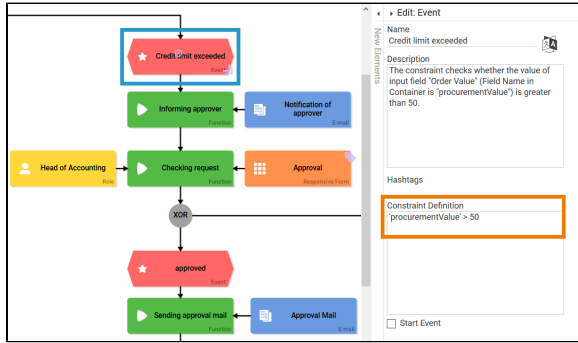
- If the demand remains in the credit limit (≤ 50), the process continues to the process step **Sending confirmation**.
- If the demand exceeds 50 Dollar, the event **Credit limit exceeded** (>50) comes into effect, triggering the next process step **Informing Approver**.

Each branch following the XOR connector requires its own event. In the element option you have to enter the corresponding condition referring to and verifying the field's value in the data container.

Verification **Credit limit adhered**

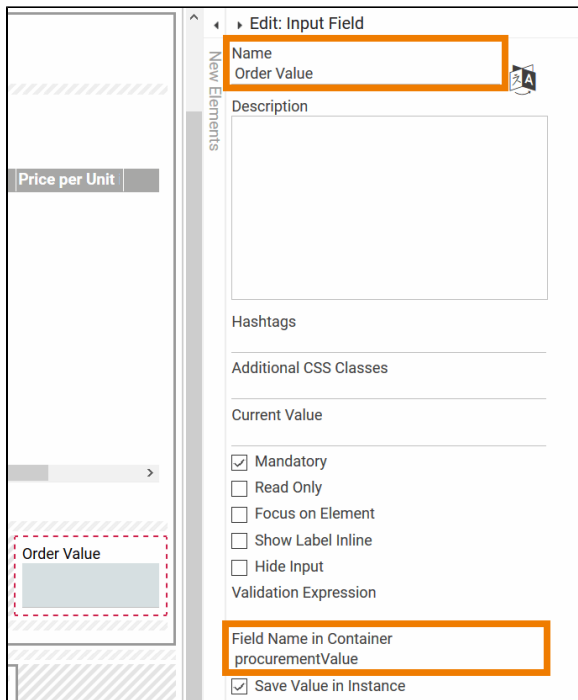


Verification **Credit limit exceeded**

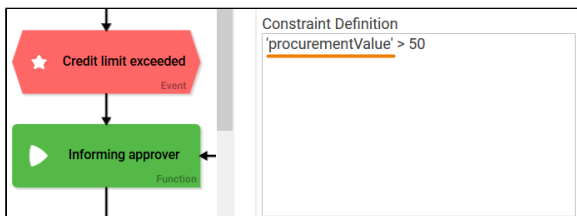


Designers can assign a **Field Name in Container** to form fields, that varies from the field's actual name (see page [The Container Principle](#) for further details). However make sure to use the **Field Name in Container** when defining the condition, since during the event verification the value in the data container will be used.

Example: The **Field Name in Container** `procurementValue` is assigned to the input field **Order Value**



Accordingly the conditions refer to `procurementValue`, the Field Name in Container:



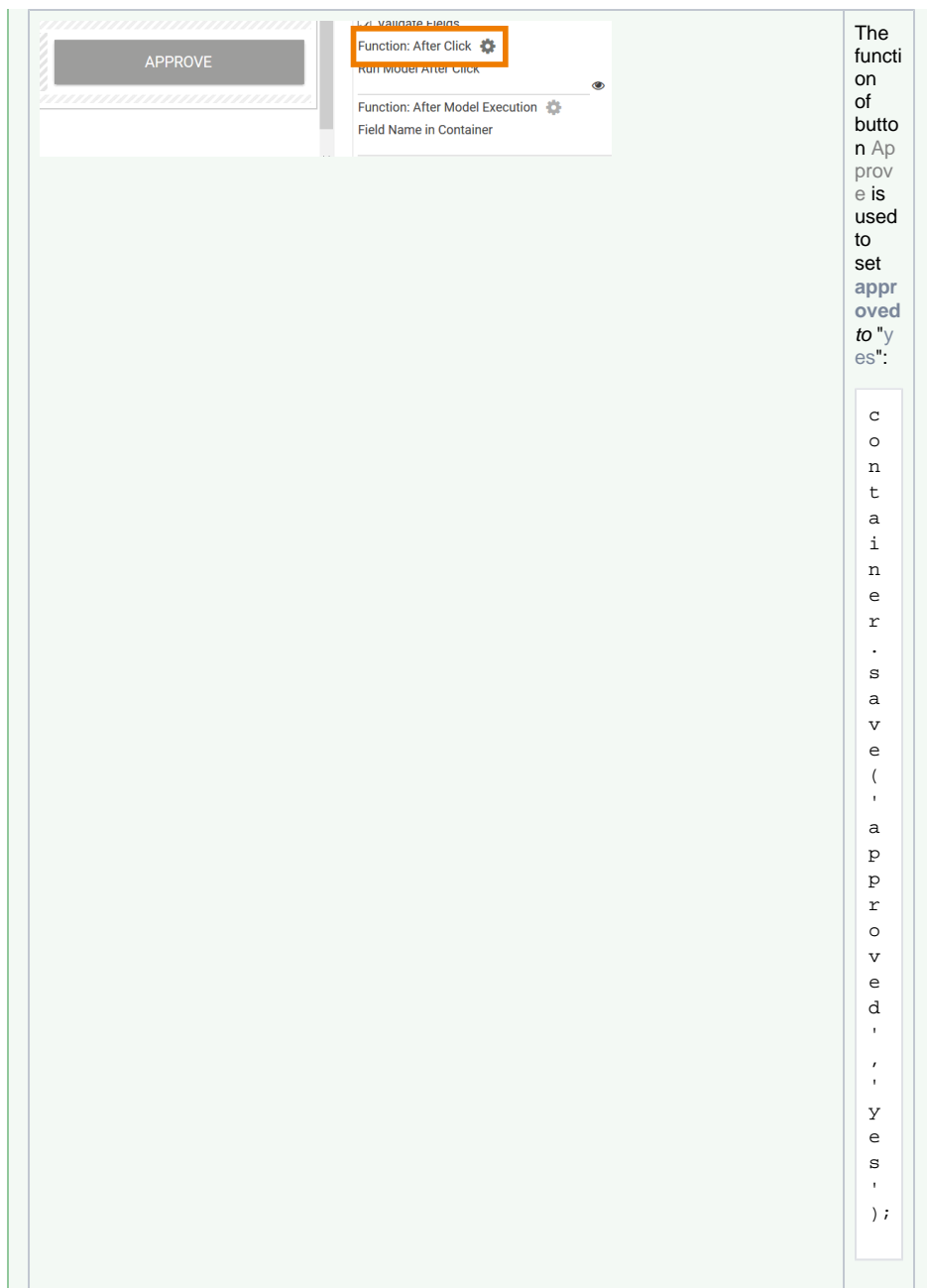
The conditions need to be defined to cover **all possible** events. In our example is not only verified if the Order Value is larger or smaller than 50 \$, but also covers the possibility that the amount equals 50 \$. Therefore all possible values are covered.

Expert Advice

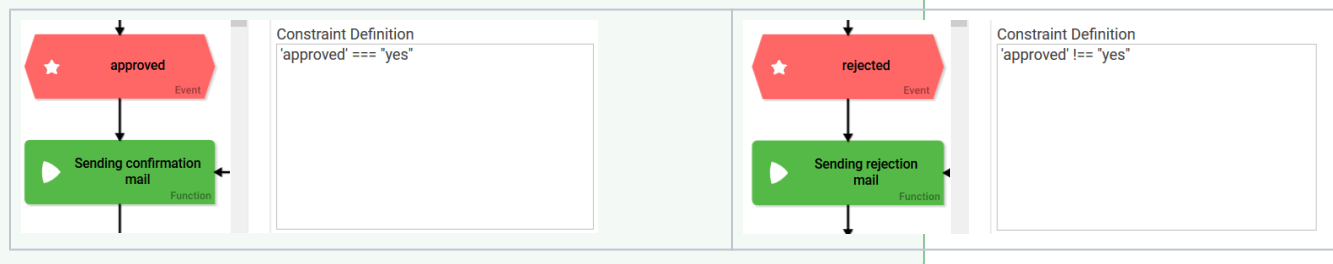
Use the **Button Function After Click** to generate variables for the event verification.

Example: Creation of variable `approved`

<div><div>REJECT</div><div>APPROVE</div></div>	<p>Insert the buttons Reject and Approve to your form.</p>
<div><div>REJECT</div><div><div>Validate Fields</div><div><div>Function: After Click</div><div>Run Model After Click</div><div>Function: After Model Execution</div><div>Field Name in Container</div></div></div></div>	<p>Set the Reject button's variable approved to 'no' in the Function: After Click:</p> <pre>container.save('approved', 'no');</pre>



Now check in the events for the variable **approved** :



Conditions: Syntax

In a condition verification **Logical Operators** are used. The placeholder needs to be encased in single quotation marks. The placeholder is the value to be referenced (= **Name** or **Field Name in Container** of the form field). String comparisons are always encased in double quotation marks.

Example:

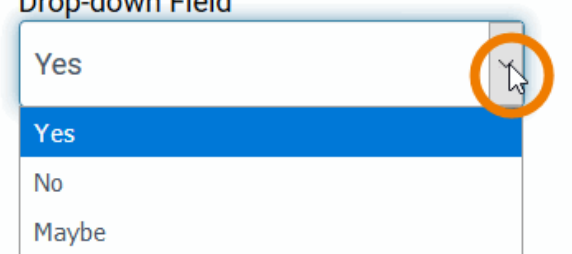
Syntax	Meaning
'Order Total' <= 150	The condition verifies if the Order Total field's value is smaller or equal to 150.
'Name' === "Meier"	The condition verifies if the entry in field Name contains the string Meier.

Expert Advice

You may also insert `container.get()` for the condition verification. In this case JavaScript conform statements need to be inserted.

Conditons: Special Cases

Some form fields have special characteristics, that need to be considered when checking conditions.

<p>Drop-down Field</p> 	<p>Drop-down Field</p> <p>The values of a Drop-down Field are always saved as value pair value/label. When checking the values of a Drop-down Field, you always need to specify if you are looking for the value entry or the label entry.</p> <p>Additional informations can be found on page Possible Values: Value and Label.</p>						
<p><input type="checkbox"/> Checkbox</p>	<p>Checkbox</p> <p>A Checkbox can only assume two values: true and false (boolean values):</p> <ul style="list-style-type: none"> • If the checkbox is activated, then the value saved in the container is true. • If it remains unchecked, then the value in the data container is false. <p>Example: Check for the MyBox checkbox' value</p> <table border="1" data-bbox="760 1291 1052 1543"> <thead> <tr> <th>Checkbox MyBox was activated</th><th>Checkbox MyBox was not activated</th></tr> </thead> <tbody> <tr> <td>'MyBox' === true</td><td>'MyBox' === false</td></tr> <tr> <td>'MyBox' !== false</td><td>'MyBox' !== true</td></tr> </tbody> </table>	Checkbox MyBox was activated	Checkbox MyBox was not activated	'MyBox' === true	'MyBox' === false	'MyBox' !== false	'MyBox' !== true
Checkbox MyBox was activated	Checkbox MyBox was not activated						
'MyBox' === true	'MyBox' === false						
'MyBox' !== false	'MyBox' !== true						
<p>Radio Button</p> <p><input checked="" type="radio"/> Blue</p> <p><input type="radio"/> Green</p> <p><input type="radio"/> Red</p> <p><input type="radio"/> Yellow</p>	<p>Radio Button</p> <p>Analog to the Drop-down Field, the value of a Radio Button is also saved as value pair value/label. When checking for the values of a Radio Button you therefore always need to specify whether you are searching for the value or the label.</p> <p>Additional informations can be found on page Possible Values: Value and Label.</p>						

Logical Operators

Find an overview of commonly used logical operators below:

Operator	Meaning
==	Equality operator
!=	Inequality operator
===	Strict equality operator
!==	Strict inequality operator
<	Smaller-than-operator
<=	Smaller-than-or-equal-operator
>	Larger-than-operator
>=	Larger-than-or-equal-operator
&&	Logical AND-operator. All connected logical expressions have to be fulfilled in order to comply with conditions.
	Logical OR-operator. At least one of the connected logical expressions has to be fulfilled in order to comply with conditions.
!	Logical NOT-operator. Reverses the logical value.



Generally always use the **strict** (in-)equality operator. This is to ensure that two operands match not only in value but also by type.