




# Modeling Tasks

Tasks are the heart of each business process - they are where the actions of the process are implemented. Actually, you can directly connect a start event to an end event and draw a process without any tasks but this would be not meaningful in most cases.

With BPMN models in **Scheer PAS Designer**, you can model the following types of tasks:

Type	Usage	Reference Documentation	Execution	State (Example)
 <b>Service Task</b>	<ul style="list-style-type: none"> <li>Task that contains implementations that are executed automatically when the process reaches the task.</li> <li>The process continues to the next element when the execution ends.</li> </ul>	<a href="#">Service Task</a>	<ul style="list-style-type: none"> <li>On Exit</li> </ul>	<b>Executed_D</b> o_something
 <b>Receive Task</b>	<ul style="list-style-type: none"> <li>Process engine waits for an external trigger message to arrive.</li> <li>Once the message has arrived, the process continues at this point.</li> </ul>	<a href="#">Receive Task</a>	<ul style="list-style-type: none"> <li>On Exit</li> </ul>	<b>Waiting_for_</b> Receive_a_ message
 <b>User Task</b>	<ul style="list-style-type: none"> <li>Process engine waits for an external trigger message to arrive when a <b>User Task</b> is reached.</li> <li>In most cases, a user form would be related to this type of task.</li> <li>The process engine continues once the form has been filled in and send back.</li> </ul>	<a href="#">User Task</a>	<ul style="list-style-type: none"> <li>Get Data</li> <li>On Exit</li> </ul>	<b>Waiting_for_</b> User_Task

## On this Page:

- [Service Tasks](#)
  - [Implement Execution](#)
    - [Error Handling](#)
- [Receive Tasks](#)
  - [Implement Execution](#)
    - [Error Handling](#)
- [User Tasks](#)
  - [Implement Execution](#)
    - [Error Handling](#)

### BPMN\_Service\_Task\_Example



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

### BPMN\_Receive\_Task\_Example



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

### 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**.

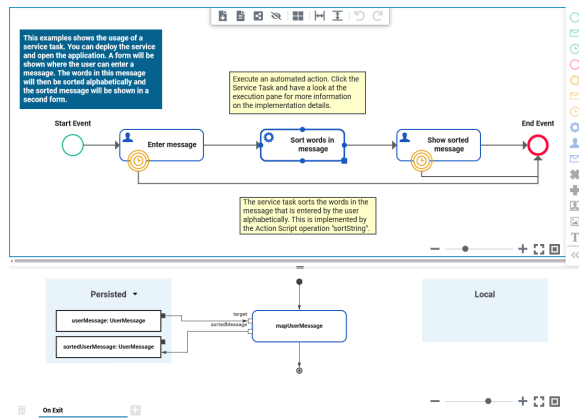
# Service Tasks

A **Service Task** contains implementations that are performed during process execution.

## BPMN\_Service\_Task\_Example



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



The screenshot above displays the BPMN diagram of the [BPMN\\_Service\\_Task\\_Example](#) service, and the execution diagram related to service task [Sort words in message](#). Although the Designer allows for service tasks without any execution diagram, this does not make much sense. Empty execution diagrams will be reported by the compiler with a warning.

Once the service task has been reached during process execution, the process state machine will switch to a state **Executed\_<name of the service task with underscores>**. In the example that would be [Executed\\_Sort\\_words\\_in\\_message](#).

## Implement Execution

For service tasks, you can add the following execution:

Execution	Description	API / Example
On Exit	Contains the execution to be performed in this process step.	No API.

You can also delete the **On Exit** execution entirely. In this case, the service task does nothing but logging that this process step has been passed.

## Error Handling

If the implemented execution is erroneous, the process goes to an error state. The process provides a retry functionality. Upon retry, the service task will be restarted from the beginning and the implemented execution will be performed once again. This means, before triggering a retry you need to fix the error.

Refer to [Persistent State Transaction Concept](#) for more information on persistent state transaction handling.

# Receive Tasks

A **Receive Task** can be used to introduce data into a process other than via a user input.

## Related Pages:

- Supported Form Elements
  - Service Task
  - Receive Task
  - User Task
- Modeling Message Reception
- Using Forms
  - Form Elements in the Data Model
  - Process Data From a Form

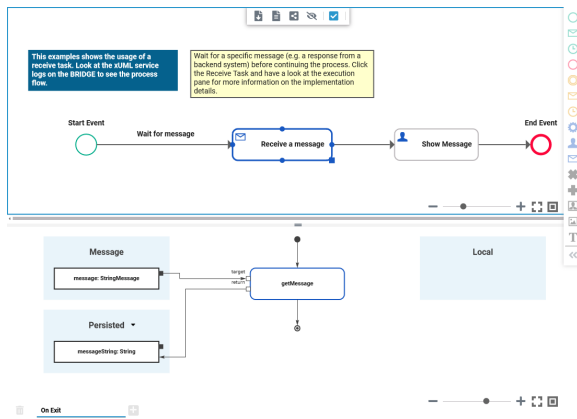
## Related Documentation:

- [Persistent State Transaction Concept](#)

## BPMN\_Receive\_Task\_Example



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



The screenshot above displays the BPMN diagram of the **BPMN\_Receive\_Task\_Example** service, and the execution diagram related to service task **Receive a Message**. Although the Designer allows for receive tasks without any execution diagram, this does not make much sense. You will at least may want to persist the incoming message to be available in the next steps of your process. Empty execution diagrams will be reported by the compiler with a warning.

Once the receive task has been reached during process execution, the process state machine will switch to a state **Waiting\_for\_<name of the receive task with underscores>**. In the example that would be **Waiting\_for\_Receive\_a\_message**.

## Implement Execution

For receive tasks, you can add the following execution:

Execution	Description	API / Example
On Exit	This execution is performed in the exit behavior of the related process state, once the message has been received. This execution gets the message that has been send as a parameter, and you can persist it to have access to the contents later on in the process. Refer to <a href="#">Modeling Message Reception</a> for more information on message handling.	<pre>POST /{id} /&lt;name of the receive task with underscores&gt; POST /{id} /Receive_a_message</pre>

Besides persisting the message, you can add more executional parts but consider the pitfalls of error handling (see below) in this case. You can also remove the **On Exit** execution entirely. In this case, the message parameter is dropped.

## Error Handling

If the implemented execution is erroneous, the process goes to an error state. The process provides a retry functionality but note the following in case of retry:

- **On Exit** is executed when the state related to the receive task (e.g. **Waiting\_for\_Receive\_a\_message**) is left. A retry will start from the process step that follows the receive task.
- The (partly erroneous) implementations of the receive task will not be processed again on retry. So consider wisely which activities to put in to the **On Exit** execution of a receive task.



Do not implement activities that e.g. rely on backend systems that may be down, or other data processing. We recommend to put these into a service task as a next process step, as error handling of service tasks differs from receive tasks.

Refer to [Persistent State Transaction Concept](#) for more information on persistent state transaction handling.

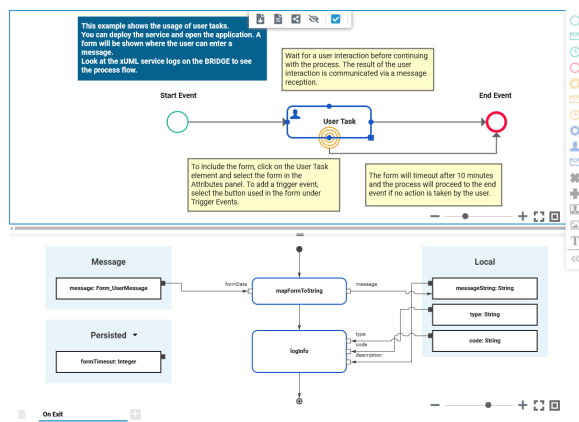
## User Tasks

Similar as with **Receive Tasks**, the process engine waits for an external trigger message to arrive when a **User Task** is reached. In most cases, a user form would be related to this type of task, and the process engine continues once the form has been filled in and send back.

**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*.



The screenshot above displays the BPMN diagram of the **BPMN\_User\_Task\_Example** service, and the execution diagram related to service step **User Task**. Although the Designer allows for user tasks without any execution diagram, this does not make much sense. You will at least may want to persist the incoming message/form data to be available in the next steps of your process. Empty execution diagrams will be reported by the compiler with a warning.

User tasks can be associated with forms. Refer to [Using Forms](#) for more information on form handling, and all other possibilities.

Once the user task has been reached during process execution, the process state machine will switch to a state **Waiting\_for\_<name of the user task with underscores>**. In the example that would be **Waiting\_for\_User\_Task**.

## Implement Execution

For user tasks, you can add the following execution:

Execution	Description	API / Example
<b>Get Data</b>	<p>This execution is performed before showing the related form. It will also be called if you restart or refresh your Browser before having sent back the form.</p> <p>You can use this execution to prepopulate form elements. It has a generated return parameter of type form class (see <a href="#">Using Forms &gt; Form Elements in the Data Model</a>). Refer to <a href="#">Using Forms &gt; Prepopulate Data Into a Form</a> for more information on form handling.</p>	<pre>GET /{id} /&lt;name of the user task with underscores&gt; GET /{id} /User_Task</pre>
<b>On Exit</b>	<p>This execution is performed in the exit behavior of the related process state, once the message containing the form data has been received. It gets the form data as a parameter, and you can persist it to have access to the contents later on in the process. Refer to <a href="#">Using Forms &gt; Process Data From a Form</a> for more information on form handling.</p>	<pre>POST /{id} /&lt;name of the user task with underscores&gt; POST /{id} /User_Task</pre>

---

Besides persisting the message, you can add more executional parts to **On Exit** but consider the pitfalls of error handling (see below) in this case. You can also remove the **On Exit** execution entirely. In this case, the message parameter is dropped.

## Error Handling

If the implemented execution is erroneous, the process goes to an error state. The process provides a retry functionality but note the following in case of retry:

- **On Exit** is executed when the state related to the user task (e.g. [Waiting\\_for\\_User\\_Task](#)) is left. A retry will start from the process step that follows the user task.
- The (partly erroneous) implementations of the user task will not be processed again on retry. So consider wisely which activities to put in to the **On Exit** execution of a user task.



Do not implement activities that e.g. rely on backend systems that may be down, or other data processing. We recommend to put these into a service task as a next process step, as error handling of service tasks differs from user tasks.

Refer to [Persistent State Transaction Concept](#) for more information on persistent state transaction handling.