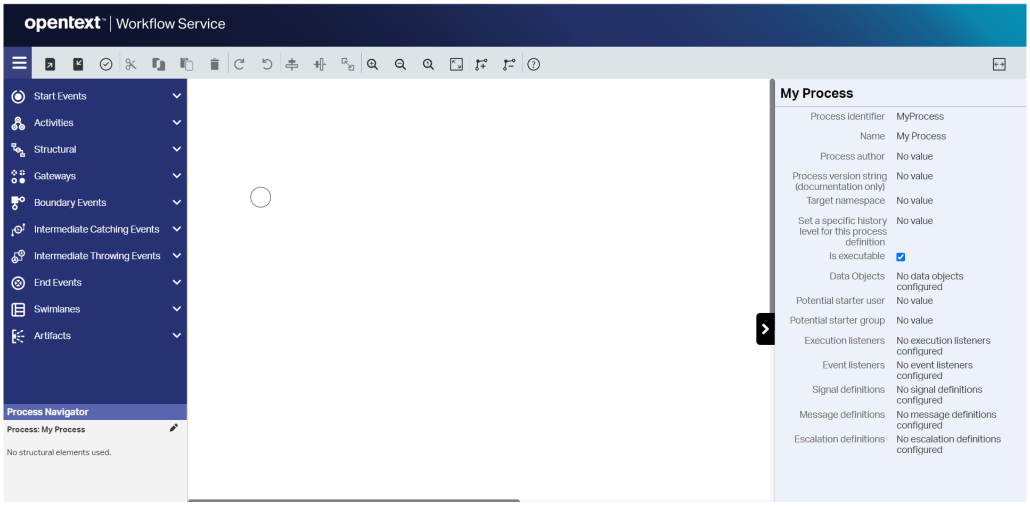
Workflow modeler constructs and properties

Workflow modeler

The workflow modeler is the primary means to create BPMN diagrams and its user interface is roughly divided into four areas: the menu bar, palette, canvas, and attribute bar.

Example



My Process

Process identifier	MyProcess
Name	My Process
Process author	No value
Process version string (documentation only)	No value
Target namespace	No value
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	No execution listeners configured
Event listeners	No event listeners configured
Signal definitions	No signal definitions configured
Message definitions	No message definitions configured
Escalation definitions	No escalation definitions configured

Processes

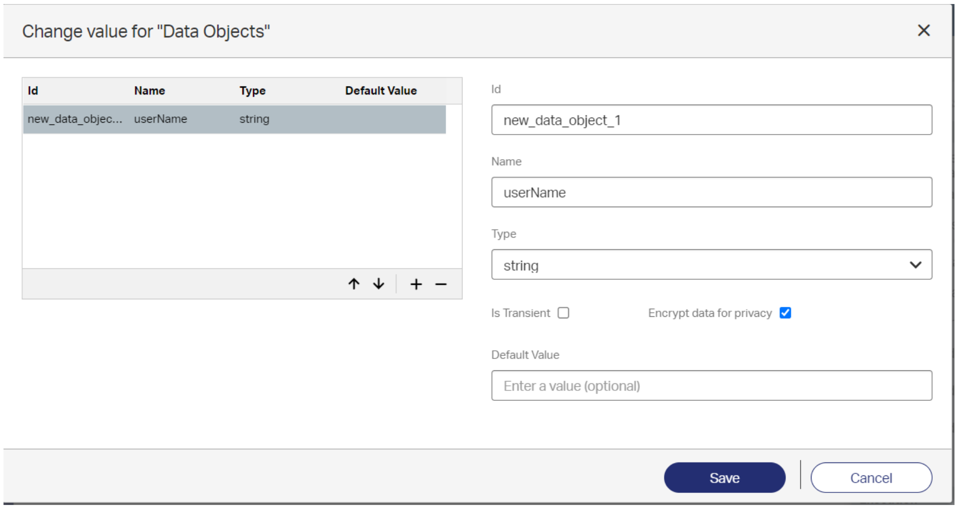
The process editor’s purpose is to model processes with elements that make up a process. The process also has a number of attributes that can be set.

Graphical notation

See the Workflow modeler example.

Attributes

Attribute	Description
Process Identifier	Unique identifier of the process.
Name	Name of the process.
Process author	Author of the process (for documentation purposes).
Process version string (documentation only)	Current version of the process.
Target namespace	Grouping of the models.
Set a specific history level for this process definition	Indicates how much chronicled data has stored for the process: None, Activity, Audit, or Full.
Asynchronous history update	Check to decide a model level configuration to update workflow-history either synchronously or asynchronously. By default, In modeler asynchro
Is executable	Decides if the process is executable. Non-executable processes just serve documentation needs and can't be started.

Data objects	Definition of data objects (metadata) available in the process. Encrypt data for privacy :  Change value for "Data Objects" Id Name Type Default Value new_data_objec... userName string Id: new_data_object_1 Name: userName Type: string Is Transient ☐ Encrypt data for privacy ☑ Default Value: Enter a value (optional) Use this option to encrypt the data used in data objects. By default, data objects are not selected for encryption.
Potential starter user	One or more users to which instantiation of this process is restricted to. A comma-separated list of users.
Potential starter group	One or more groups to which instantiation of this process is restricted to. Only users who are part of atleast one of these groups can create an ins A comma-separated list of groups.
Execution listeners	Active execution listeners will respond to the following events occurred on a process: • Start: Happens when the process instance starts. • take: Happens when the process instance transition is taken • End : Happens when the process instance completes.
Event listeners	Event listeners of this process that can react to many different events.
Signal definitions	Definition of all signals used in the process.
Message definitions	Definition of all messages used in the process.
Escalation definitions	Definition of all escalations used in the process.

Event listeners

Workflow modeler provides a way to handle below events of a process, and to send mail notification when these events occur during process execution.

Events	Mail Notification

Activity ACTIVITY_CANCELLED ACTIVITY_COMPENSATE ACTIVITY_COMPLETED ACTIVITY_ERROR_RECEIVED ACTIVITY_MESSAGE_CANCELLED ACTIVITY_MESSAGE_RECEIVED ACTIVITY_MESSAGE_WAITING ACTIVITY_SIGNALED ACTIVITY_SIGNAL_WAITING ACTIVITY_STARTED	Yes
Process PROCESS_CANCELLED PROCESS_COMPLETED PROCESS_COMPLETED_WITH_TERMINATE_END_EVENT PROCESS_COMPLETED_WITH_ERROR_END_EVENT PROCESS_CREATED PROCESS_STARTED	Yes
Task TASK_ASSIGNED TASK_COMPLETED TASK_CREATED	Yes

Mail notification takes below attributes.

Attribute	Description
To	The receivers of the email. Multiple emails can be added using comma-separated list.
From	The email address of sender. If not given, the default provided "from address" is used.
CC	The CC (Carbon Copy) email recipients. Multiple emails can be added using comma-separated list.
BCC	The BCC (Blind Carbon Copy) email recipients. Multiple emails can be added using comma-separated list.
Subject	The subject of the email.
Mail text	Email text. If at receiver's end, HTML is not supported then email is displayed in text format.
Headers	Mail headers
Text variable	Process instance variable
HTML	Email content in HTML format. This allows for creation of rich formatting and usage of images.
HTML variable	Process instance variable
Charset	Character set to be used in the email.

Execution listeners

Execution listeners allows user to execute a custom action when certain events occur during process execution. The events that can be captured are:

- Start: Configures the listener when a process instance starts
- take: Configures the listener when a process instance transition is done

- End : Configures the listener when a process instance completes

For each event, below actions can be configured.

- Execute an expression

  Given expression is evaluated based on the process instance data.

- Send a notification

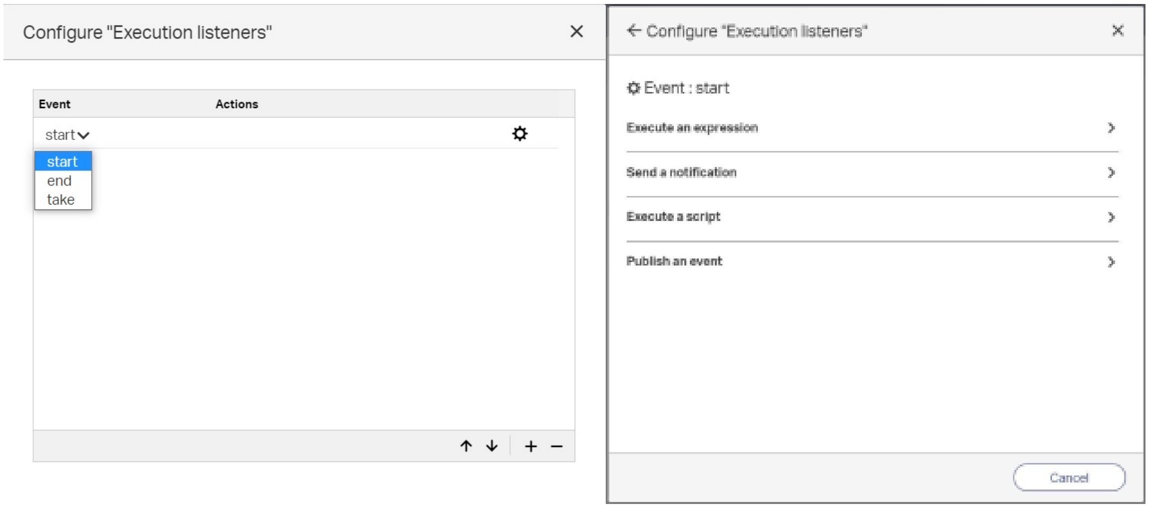
A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity.

- Execute a script

A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.

- Publish an event

Configure this option to publish an event to OT2 event service for a specific event type. 'Event key' is the event type Id which is part of OT2 event service. 'Event data' is the application data expected in JSON format. Application data can be published to event server as described in Workflow Listeners data Integration with OT2 Event Service



Expressions

Unified Expression Language (UEL) is used by Workflow for expression-resolving. In Conditional sequence flows, Task Listeners, Execution Listeners and Java Service tasks these expressions are used.

Value expression

This expressions resolves to a value. By default, all process variables are available to use.

Some examples :
${myVar}

Apart from all process variables, there are a some default objects which are available that can be used in these expressions:

execution: The DelegateExecution contains more information about current execution. Execution can be used to create process global variables from execution listeners.

task: The DelegateTask contains more information about current Task. Task can be used to create task local variables from task listeners.

Some examples for value expressions are :

Creating Process Variables :
${execution.setVariable(String VariableName , Object Value)}

Creating Task Variables :
${task.setVariableLocal(String VariableName , Object Value , boolean Flag)}

Retrieving Process Variable Value :
${execution.getVariable(String VariableName)}

Retrieve Task Variable Value :

${variables:get(String VariableName)}

Null Checking:
${VariableName == null}

Checking Value for a Variable :
${VariableValue == Value}

Conditional Statement :

${Variable Logical_Operator Value? Expression to be evaluated if true : Expression to be evaluated if false)}

Example : ${Variable >= 10 ? execution.setVariable("Permission","accepted") : execution.setVariable("Permission","rejected")}

Expression functions

Under the variables namespace a set of out-of-the-box functions are available, To make working with process variables easy.

variables:get(varName) :

Retrieves the value of a variable. If we use variable name directly in the expression and if that variable doesn't exist then we throw an exception but if we use get function then we won't throw any exception even if the variable doesn't exist. For example ${varName == "hello"} would throw an exception if varName doesn't exist, but ${var:get(varName) == 'hello'} will just work.

variables:getOrDefault(varName, defaultValue) :

Similar to get, but here we can initialize variable with a default value, returns default value when the value is null or variable isn't set.

variables:exists(varName) :

Returns true if the variable has a non-null value.

variables:isEmpty(varName) (alias :empty) :

Checks if value is not empty. If variables are of type String then they are said to be empty if string is empty. If variable is null, always true is returned.

variables:isNotEmpty(varName) (alias :notEmpty) :

The reverse operation of isEmpty.

variables:equals(varName, value) (alias :eq) :

Checks if a given value is equal to the variable. This function is a shorthand for the expression ${execution.getVariable("varName") != null && execution.getVariable("varName") == value}.

If the variable value is null, false is returned (unless compared to null).
variables:notEquals(varName, value) (alias :ne) :

The reverse comparison of equals.

variables:contains(varName, value1, value2, …) :

Checks if all values provided are contained within a variable. If variables are of type String, then passed values are used as substrings that need to be part of the variable.

variables:containsAny(varName, value1, value2, …) :

Same as contains function, It returns true even if any one of the passed values are present in the variable value.

variables:base64(varName) :

Converts a Binary or String Variable in Base64 String.

variables:lowerThan(varName, value) (alias :lessThan or :lt) :

Shorthand for ${execution.getVariable("varName") != null && execution.getVariable("varName") < value}

variables:lowerThanOrEquals(varName, value) (alias :lessThanOrEquals or :lte) :

Similar, but now for < =

variables:greaterThan(varName, value) (alias :gt) :

Similar, but now for >

variables:greaterThanOrEquals(varName, value) (alias :gte) :

Similar, but now for > =

Note: The variables namespace is aliased to vars or var. So variables:get(varName) is equivalent to writing vars:get(varName) or var:get(varName).

Retrieving and Updating task properties through task listeners

Using expressions we can either retrieve or update task properties like name, priority, description, due date, assignee, etc of a task in the task listeners. Below are some examples of how to retrieve and update some of the task properties.

Task Name :

The task name can be retrieved using the below expression.

${task.getName()}

The task name can be updated using the below expression.

${task.setName("Sample Task")}

Task Priority :

The task priority can be retrieved using the below expression.

${task.getPriority()}

The task priority can be updated using the below expression.

${task.setPriority(10)}

Task Description :

The task description can be retrieved using the below expression.

${task.getDescription()}

The task description can be updated using the below expression.

${task.setDescription("Sample Description")}

Task Assignee :

The task assignee email can be retrieved using the below expression.

${task.getAssignee()}

The task assignee otds uuid can be retrieved using the below expression.

${task.getAssigneeUuid()}

The task assignee can be updated using the below expression.

${task.setAssignee("sampleuser@opentext.com")} or ${task.setAssignee("d91e60a2-1fed-4c84-96a2-1ff117fbe2dd")}

Task DueDate :

The task due date can be retrieved using the below expression.

${task.getDueDate()}

The task due date can be updated using the below expression.

${task.setDueDate(2020-09-22T03:25:18.108Z)}

Task Id :

The task id can be retrieved using the below expression.

${task.getId()}

Process Instance Id :

The task process instance id can be retrieved using the below expression.

${task.getProcessInstanceId()}

Retrieving and Updating process Instance properties through execution listeners

The process instance properties can be retrieved and updated using the expressions in execution listeners, these expressions can be used on process executions listeners and not on task.

Process Instance Id :

The process instance id can be retrieved using the below expression.

${execution.getProcessInstanceId()}

Process Instance Name :

The process instance name can be retrieved using the below expression.

${execution.getName()}

The process instance name can be updated using the below expression.

${execution.setName("Sample Process Name")}

Process Start User Id :

The process start user email can be retrieved using the below expression.

${execution.getStartUserId()}

The process start user otds uuid can be retrieved using the below expression.

${execution.getStartUserUuid()}

The process start user id can be updated using the below expression.

${execution.setStartUserId("exampleuser@opentext.com")} or ${execution.setStartUserId("d91e60a2-1fed-4c84-96a2-1ff117fbe2dd")}

Process Business key :

The process business key can be retrieved using the below expression.

${execution.getBusinessKey()}

The process business key can be updated using the below expression.

${execution.setBusinessKey("Sample Business Key")}

Process Definition Key :

The process definition key can be retrieved using the below expression.

${execution.getProcessDefinitionKey()}

The process definition key can be updated using the below expression.

${execution.setProcessDefinitionKey("Sample Process Definition key")}

Process Definition Id :

The process definition id can be retrieved using the below expression.

${execution.getProcessDefinitionId()}

Process Definition Name :

The process definition name can be retrieved using the below expression.

${execution.getProcessDefinitionName()}

Construct properties

Many constructs are available to create a business process and are arranged into logical groups. Selecting an item on the canvas displays a list of properties to configure.

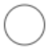
Start events

A start event specifies where a process initiates. The type of start event defines how the process starts, for example, on arrival of a message or at a specific time interval.

Start event

A start event implies that the trigger to start the process instance is unspecified and the business process engine cannot anticipate when to start it. A start event triggers a process instance by an API call to any of the startProcessInstanceByXXX functions.

Graphical notation



Attributes

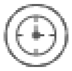
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Initiator	Variable name containing authenticated user ID is stored when the process is triggered.
Form Properties	Sets the form properties.
Form Key	Form related URLs to use in applications.
Validate form fields (server-side)	If validate form fields expression evaluates to true, once the form is submitted then fields of the form are validated as per form model restrictions.

Start timer event

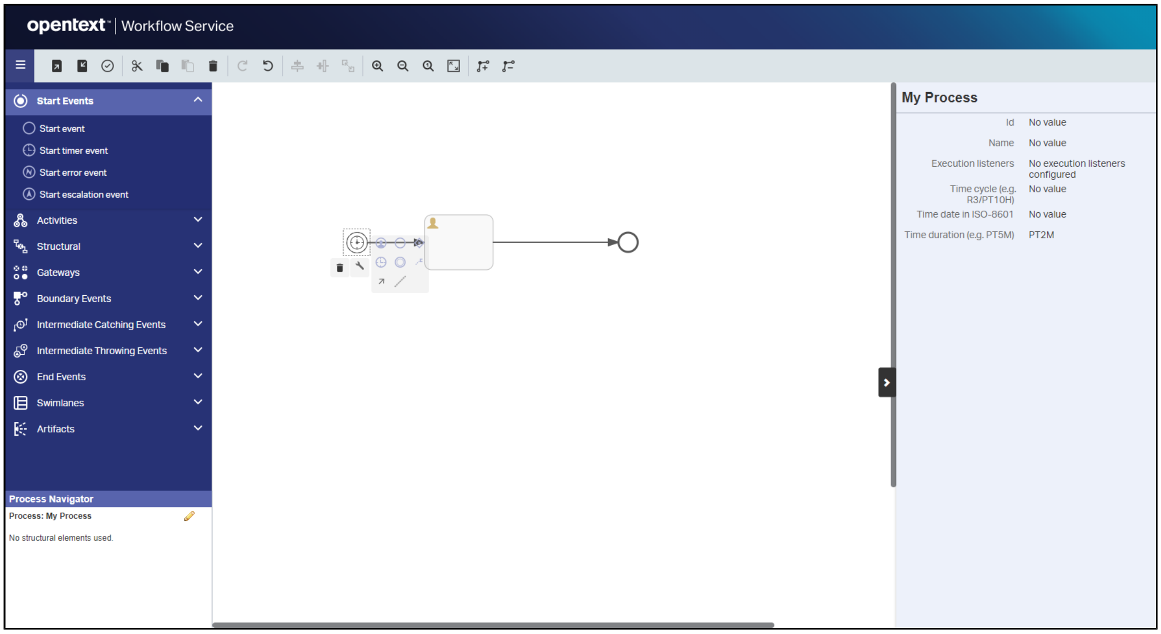
A start timer event creates a process instance at a specific time. It can be used for processes that must start only once and at specific time intervals.

Note: A subprocess cannot have a start timer event.

Graphical notation



Example



My Process

Id	No value
Name	No value
Execution listeners	No execution listeners configured
Time cycle (e.g. R3/PT10H)	No value
Time date in ISO-8601	No value
Time duration (e.g. PT5M)	PT2M

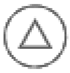
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Time cycle (e.g. R3 /PT10H)	Defines a repeating time interval, This is used to send multiple reminders for a delayed user task or to start the process periodically. Time cycle component can be in repeating time duration format as defined by the ISO 8601 standard. example, three repeating time intervals, lasting 10 hours each. We can also use corn expression e.g 0 0/5 * * * ? shows trigger firing every 5 minutes, starting at the full hour. We can also provide the end date as an attribute to the time cycle such as <timeCycle flowable:endDate="2015-02-25T16:42:11+00:00">R3 /PT10H</timeCycle>
Time date in ISO-8601	Specifies a fixed date (ISO 8601 format) when the trigger will fire.
Time duration (e.g. PT5M)	Specifies how long the timer must run before it is fired. A `timeDuration` can be defined as a sub-component of `timerEventDefinition`. The ISO 8601 format is used as required by the BPMN 2.0 specification.

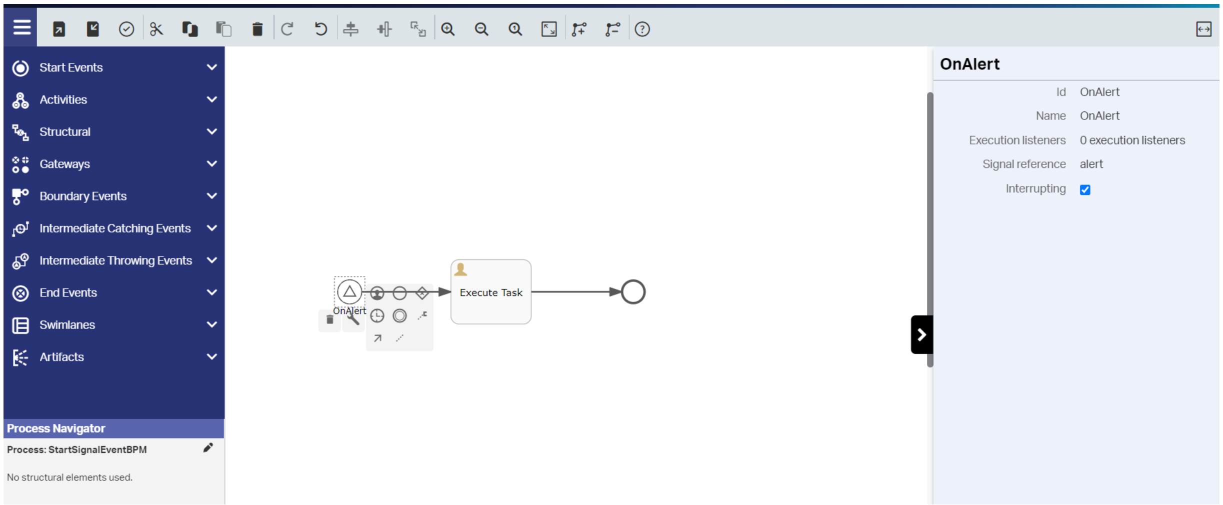
Start signal event

Start signal event triggers a process instance by using named signal. This signal can be fired within a process instance through the API or using intermediary signal throw event. In these both cases, all the process definitions which have signal start event with same name will be started.

Graphical notation



Example



Process: StartSignalEventBPM

No structural elements used.

OnAlert

Id	OnAlert
Name	OnAlert
Execution listeners	0 execution listeners
Signal reference	alert
Interrupting	☑

Perform the following steps:

1. Create a process model(**Process1**) with start signal event.
2. In the process attributes 'signal definitions' create a required named signal (eg: '**alert**').
3. In the 'signal event' attributes click on the 'Signal reference' drop-down menu and select the created named signal.
4. Similarly, create another process model(**Process2**)
5. In the process attributes 'signal definitions' create a required named signal (eg: '**alert**').
6. Add 'Intermediate signal throwing event' to the model, In the 'signal event' attributes click on the 'Signal reference' drop-down menu and select the created named signal.
7. Deploy (save and publish) both the models.
8. Now trigger **Process2** process. This process instance will throw the '**alert**' event globally.
9. Then **Process1** model start signal event receives this signal and it initiates the **Process1** process.

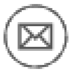
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Signal reference	Provide the signal name
Interrupting	Check this option to terminate all parent execution

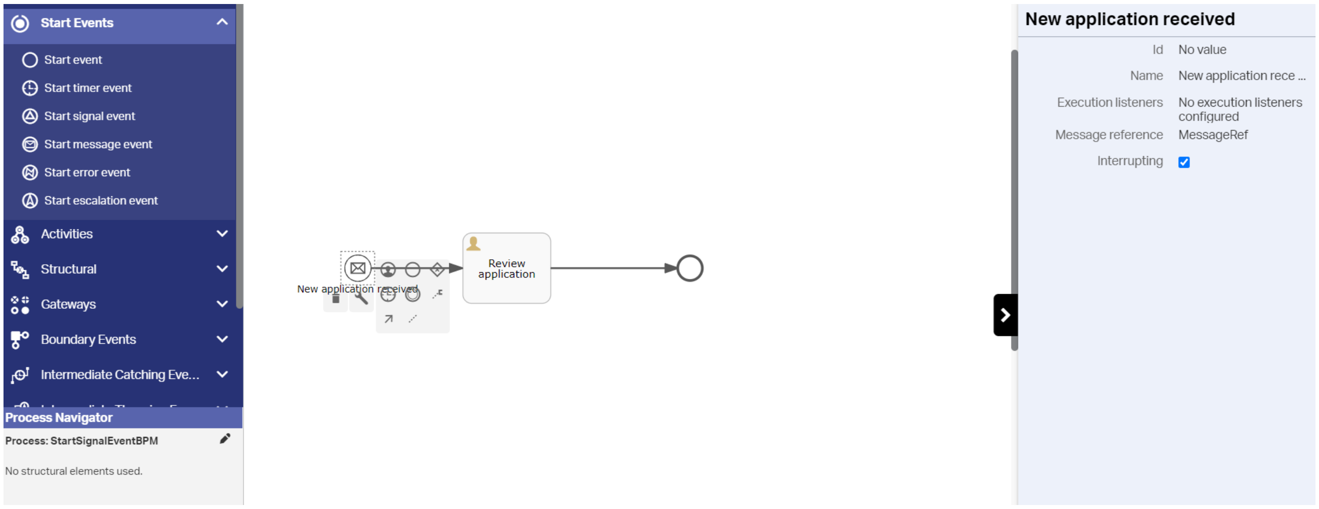
Start message event

Message start event triggers a process instance by using named message. This makes us to choose the correct start event from a set of other alternative start events by using message name. Message start event name must be unique over all deployed process definitions. Multiple message start events on a process definition with the same name are not allowed.

Graphical notation



Example



New application received

Id	No value
Name	New application rece ...
Execution listeners	No execution listeners configured
Message reference	MessageRef
Interrupting	☑

Perform the following steps:

1. Create a process model with start message event.
2. In the process attributes 'message definitions' create a required named message (eg: 'MessageRef').
3. In the 'message event' attributes click on the 'Message reference' drop-down menu and select the created named message.
4. Deploy (save and publish) the model.
5. Execute workflow rest API '/runtime/process-instances' with the given message name 'MessageRef' to initiate the above process as below in "body"

   Example request body:

   ```
   {
        "message" : "MessageRef"
   }
   ```

Attributes

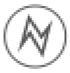
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.

Message reference	Provide the message name
Interrupting	Check this option to terminate all parent execution

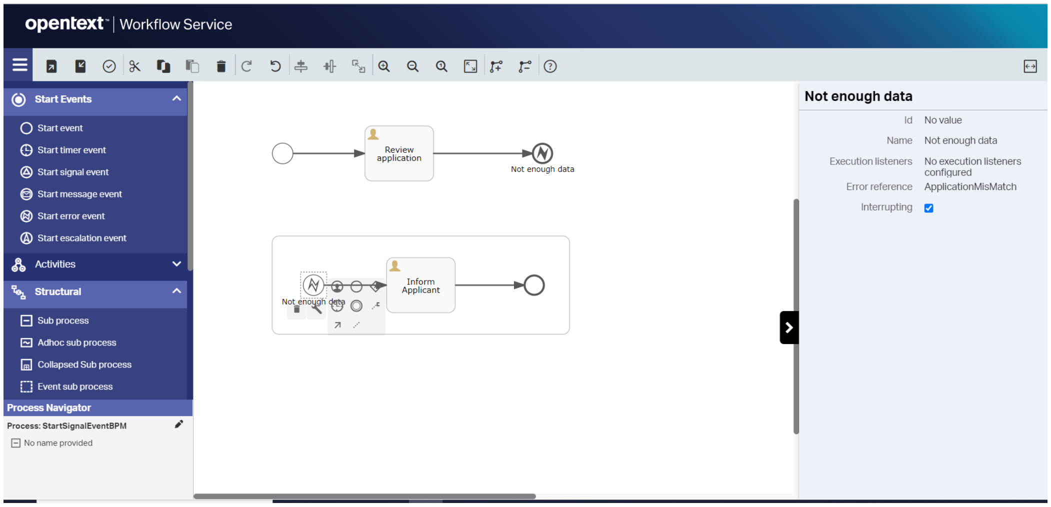
Start error event

An start error event can trigger an event sub-process but not a process instance.

Graphical notation



Example



Not enough data

Id	No value
Name	Not enough data
Execution listeners	No execution listeners configured
Error reference	ApplicationMisMatch
Interrupting	☑

Perform the following steps:

1. Create a process model with an End error event.
2. Type an error name in Error reference.
3. Add the Event sub-process with a Start error event to the model.
4. Similarly, for the Start error event, type the same error reference.
5. Save and publish the model.
6. Trigger the process. The End error event throws an error. The error event is captured by the Start error event and initiates the sub-process.

Attributes

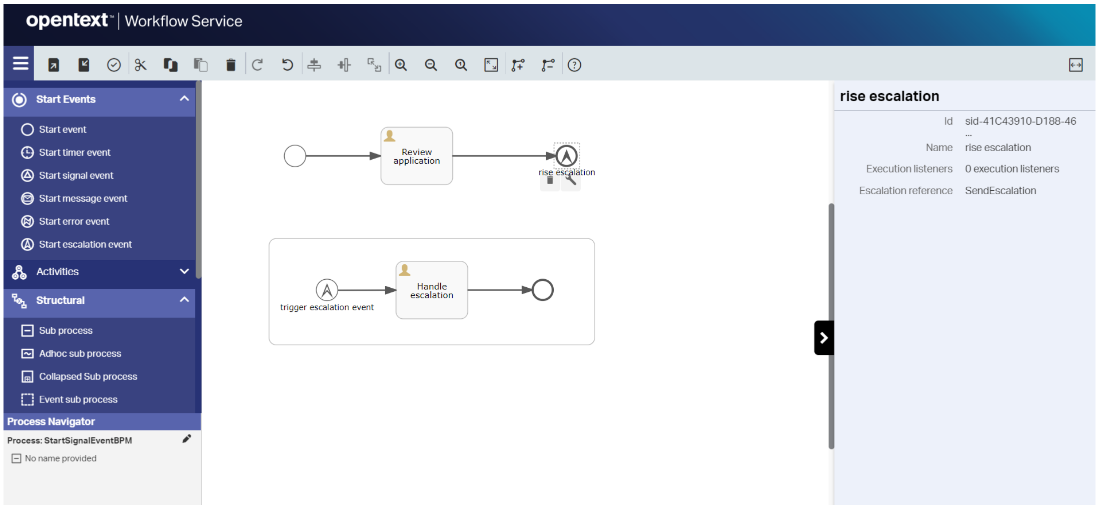
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Error reference	Name of the error.
Interrupting	Terminates all parent execution, if selected.

Start escalation event

An escalation start event can trigger an event sub-process but not a process instance. Unlike an error, an escalation event is non-critical and execution continues at the point where the error is thrown.

Graphical notation

Example

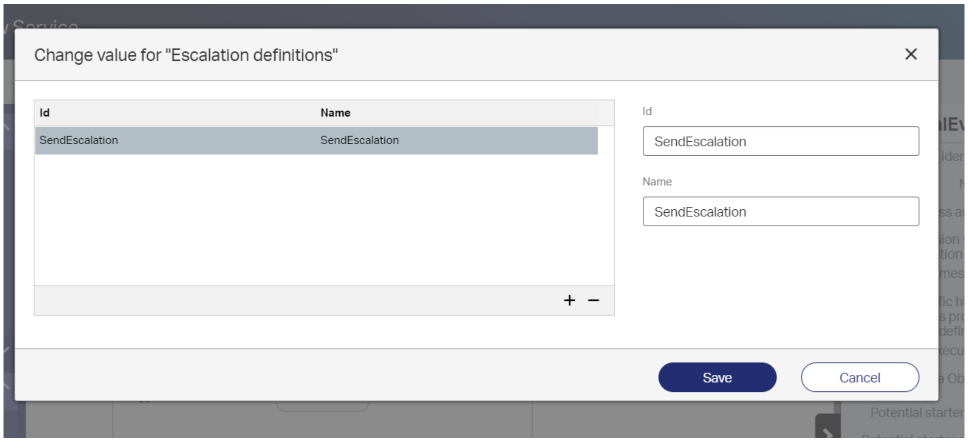


rise escalation

Id	sid-41C43910-D188-46 ...
Name	rise escalation
Execution listeners	0 execution listeners
Escalation reference	SendEscalation

Perform the following steps:

1. Create a process model and add the required named escalation in the escalation definition in the process attributes.



Change value for "Escalation definitions"

Id	Name
SendEscalation	SendEscalation

Id: SendEscalation

Name: SendEscalation

2. Add an Intermediate escalation throwing event or an End escalation event to the model and select the created escalation reference.
3. Add the Event sub-process with the Start escalation event to the model.
4. Similarly, for the Start escalation event, add the same error reference.
5. Save and publish the model.
6. Trigger this process. The Intermediate escalation event or End escalation event throws an escalation. This escalation event is captured by the Start error event and initiates the sub-process.

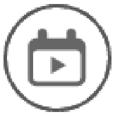
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Escalation reference	Name of the escalation.
Interrupting	Terminates all parent execution, if selected.

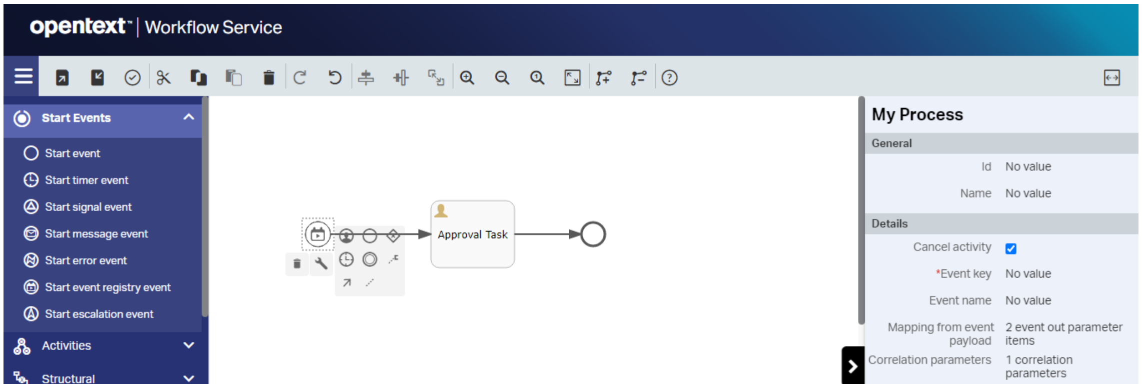
Start event registry event

A start registry event creates a process instance with an incoming event, along with a correlation. This means when an appropriate event is received through the event registry, all the process definitions which have start registry event with the same event key will be started. When the event is received, the process instance is created asynchronously. Any errors in the process execution can be handled and recovered as deadletter jobs.

Graphical notation



Example



My Process

General

Id	No value
Name	No value

Details

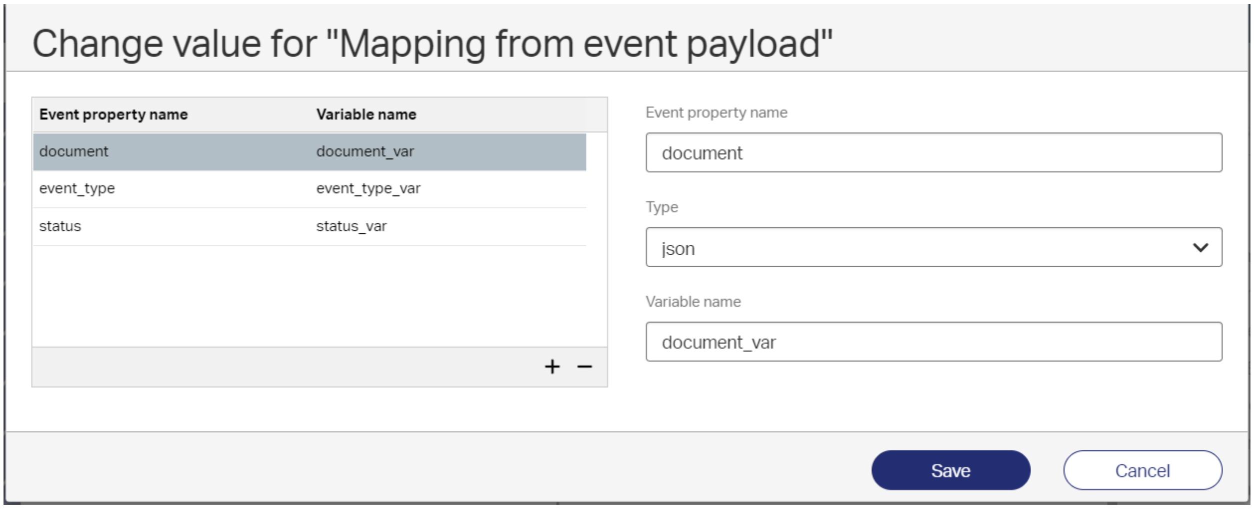
Cancel activity	☑
*Event key	No value
Event name	No value
Mapping from event payload	2 event out parameter items
Correlation parameters	1 correlation parameters

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Event Key	Key of the deployed event definition.
Event name	Name of the event definition.
Mapping from event payload	The required event payload values can be mapped to process variables.
Correlation parameters	When multiple process definitions are deployed, then the incoming event message can be matched against one of the correlation parameter.

Mapping from event payload :

Parameters can be configured to create required process variables from the event payload and can be used anywhere in the execution.



Change value for "Mapping from event payload"

Event property name	Variable name
document	document_var
event_type	event_type_var
status	status_var

Event property name: document

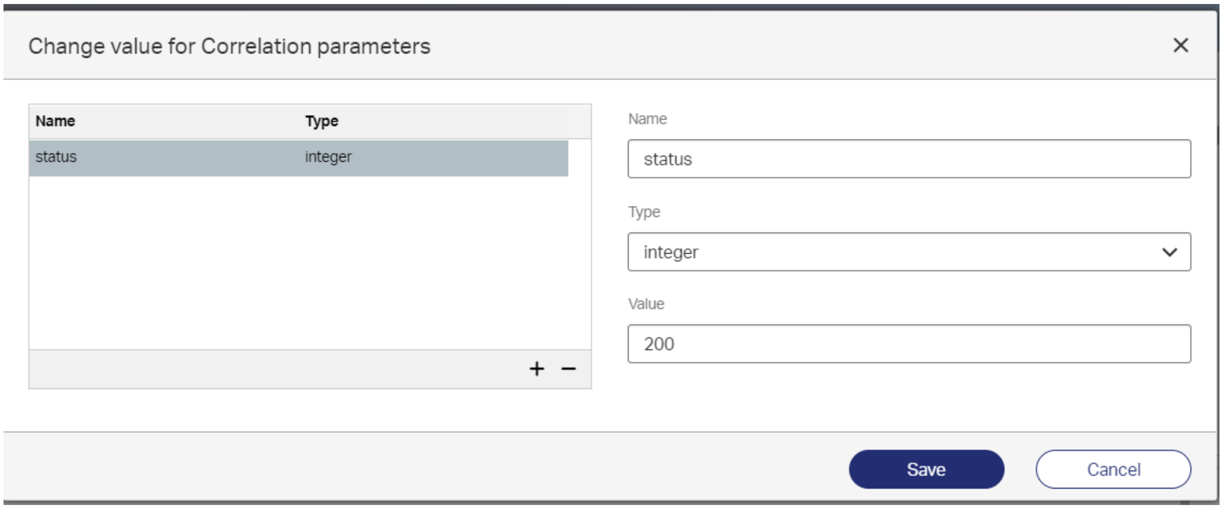
Type: json

Variable name: document_var

In the above example, three event parameters i.e document, event_type and status from event payload will be mapped to process variables document_var, event_type_var and status_var.

Correlation parameters :

Parameters can be configured to match the values against event payload, we could match multiple correlation parameters, if all of the correlation parameters are matched with the values of the received event payload then only that process definition process instant will get created.



Change value for Correlation parameters

Name	Type
status	integer

Name: status

Type: integer

Value: 200

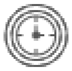
In the above example, one correlation parameter i.e status needs to match exactly against the status of the received event payload.

Intermediate catching events

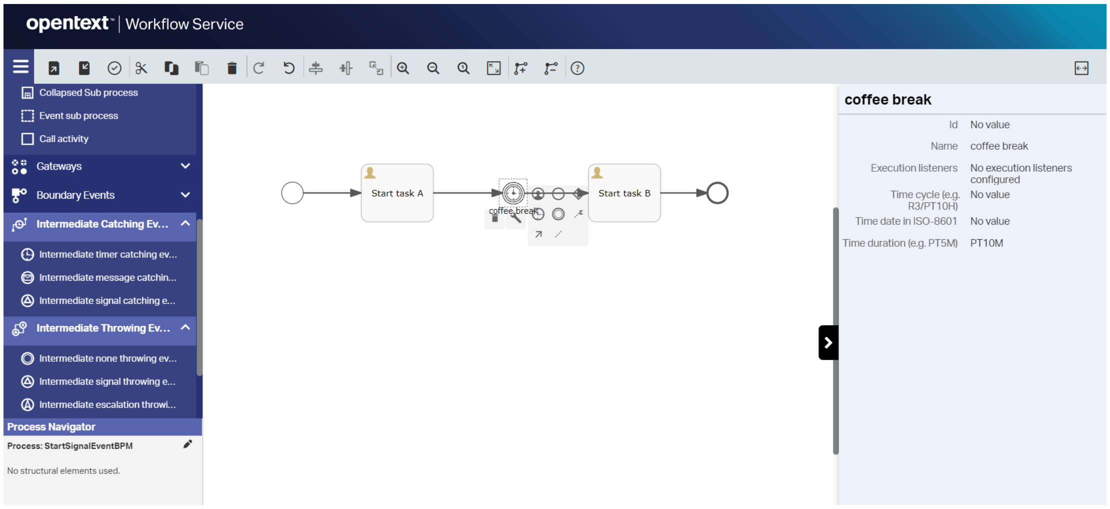
Intermediate timer catching event

Intermediate timer event behaves as a stopwatch. Timer starts when an execution comes at catching event activity. When the timer is fired (after some specified interval), the sequence flow going out of the intermediate timer event is followed.

Graphical notation



Example



opentext™ | Workflow Service
Collapsed Sub process
Event sub process
Call activity
Gateways
Boundary Events
Intermediate Catching Ev...
Intermediate timer catching ev...
Intermediate message catchin...
Intermediate signal catching e...
Intermediate Throwing Ev...
Intermediate none throwing ev...
Intermediate signal throwing e...
Intermediate escalation throwi...
Process Navigator
Process: StartSignalEventBPM
No structural elements used.
Start task A
coffee break
Start task B

coffee break

Id	No value
Name	coffee break
Execution listeners	No execution listeners configured
Time cycle (e.g. R3/PT10H)	No value
Time date in ISO-8601	No value
Time duration (e.g. PT5M)	PT10M

When the process executes, after the first activity completes, the timer waits for 10 minute (PT1M) to process the next activity.

Attributes

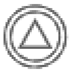
Attribute	Description

ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Time cycle (e.g. R3 /PT10H)	Defines a repeating time interval, This is used to send multiple reminders for a delayed user task or to start the process periodically. Time cycle component can be in repeating time duration format as defined by the ISO 8601 standard. example, three repeating time intervals, lasting 10 hours each.
Time date in ISO-8601	Specifies a fixed date (ISO 8601 format) when the trigger will fire.
Time duration (e. g. PT5M)	Specifies how long the timer must run before it is fired. A `timeDuration` can be defined as a sub-component of `timerEventDefinition` . The ISO 8601 format is used as required by the BPMN 2.0 specification.

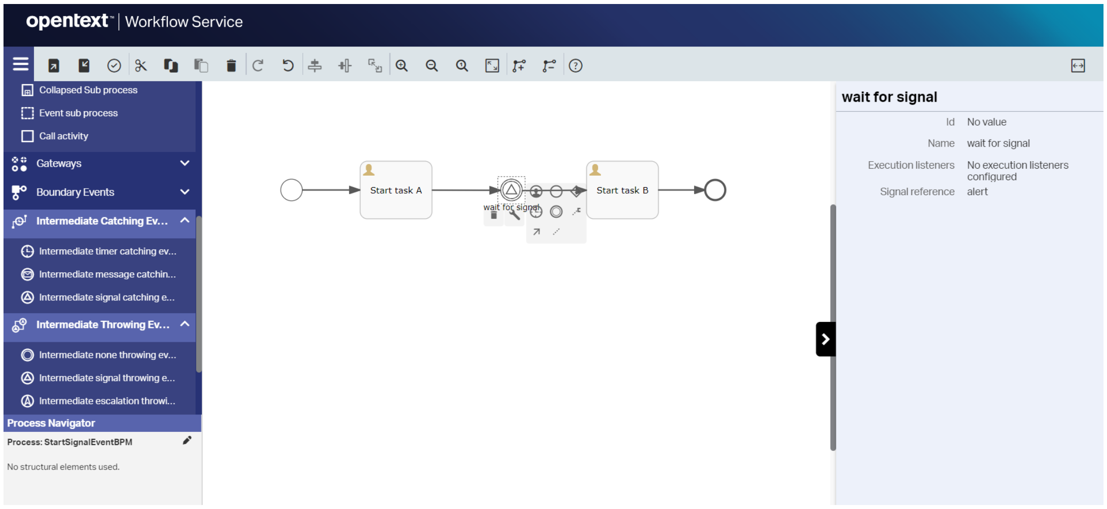
Intermediate signal catching event

Intermediate signal catching event is used to catch signals having same signal name as referenced signal definition. If two signal events are active and catching the same signal event, even though they are part of different process instances both events will be triggered.

Graphical notation



Example


opentext | Workflow Service
Collapsed Sub process
Event sub process
Call activity
Gateways
Boundary Events
Intermediate Catching Ev...
Intermediate timer catching ev...
Intermediate message catchin...
Intermediate signal catching e...
Intermediate Throwing Ev...
Intermediate none throwing ev...
Intermediate signal throwing e...
Intermediate escalation throwi...
Process Navigator
Start task A
wait for signal
Start task B

Process: StartSignalEventBPM

No structural elements used.

wait for signal

Id	No value
Name	wait for signal
Execution listeners	No execution listeners configured
Signal reference	alert

When the process executes, after the first activity completes, the signal event waits for the `alert` named signal event to process the next activity. The signal can be fired within a process instance using the API discussed in the start signal event or through intermediary signal throw event.

To trigger the signal event on a specific process instance, use the rest API `/runtime/executions/{executionId}`. Pass the signal name in the request body.

Example request body

```
{
  "action":"signalEventReceived",
  "signalName":"alert",
  "variables": [  ]
}
```

To obtain the required process instance `executionId`, use the rest API `POST /query/executions`.

Example request body

```
{
        "processInstanceId": "72fb2ddf-1060-11ea-ba8c-3ce1a14eadub",
        "signalEventSubscriptionName": "alert"
}
```

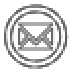
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Signal reference	Name of the signal.

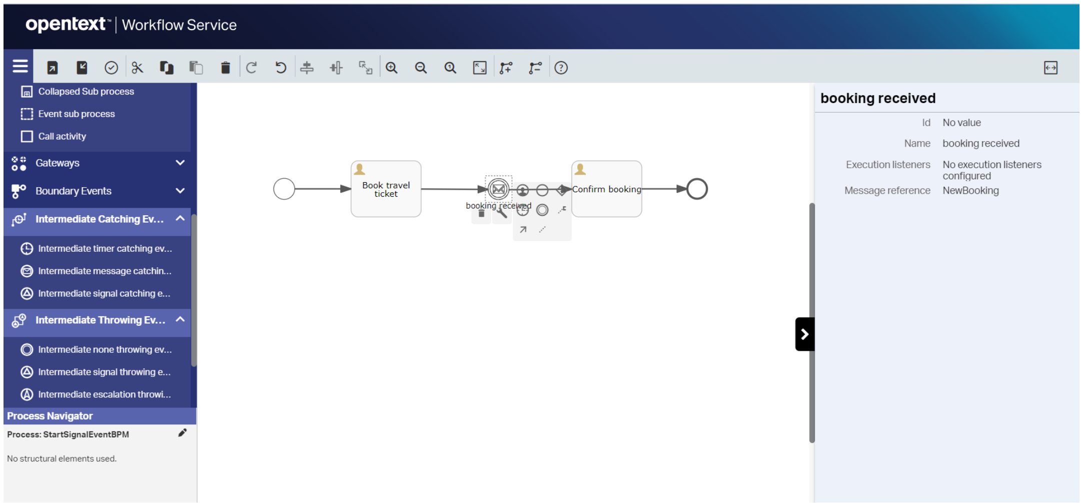
Intermediate message catching event

Intermediate catching message is used to event catches messages with a specified name.

Graphical notation



Example


opentext | Workflow Service
Collapsed Sub process
Event sub process
Call activity
Gateways
Boundary Events
Intermediate Catching Ev...
Intermediate timer catching ev...
Intermediate message catchin...
Intermediate signal catching e...
Intermediate Throwing Ev...
Intermediate none throwing ev...
Intermediate signal throwing e...
Intermediate escalation throwi...
Process Navigator
Process: StartSignalEventBPM
No structural elements used.
Book travel ticket
booking received
Confirm booking

booking received

Id	No value
Name	booking received
Execution listeners	No execution listeners configured
Message reference	NewBooking

When we execute the above process, After completion of first activity message event wait for the 'NewBooking' named message to process the next activity.

- To trigger the message event use the rest API '/runtime/executions/{executionId}', pass the message name in the request body.

  Example request body

  ```
  {
          "action": "messageEventReceived",
          "messageName": "NewBooking",
          "variable": []
  }
  ```

- To get the required process instance executionId, use the rest API 'POST /query/executions

  Example request body

```
{
    "processInstanceId": "72fb2ddf-1060-11ea-ba8c-3ce1a14eadfb",
    "messageEventSubscriptionName": "NewBooking"
}
```

Attributes

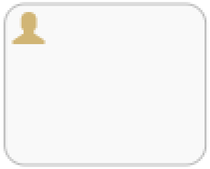
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Message reference	Provide the message name

Activities

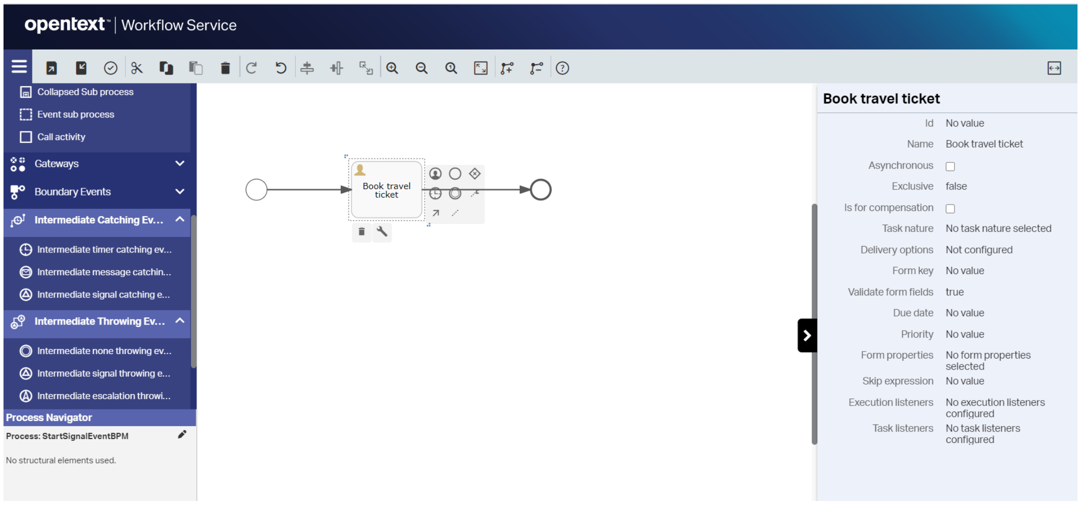
User task

A user task is a common workflow task where humans perform an action with the help of a software application. Task list manager schedules a user task. In a workflow, the main way of interacting with humans involved in a process is through user tasks. When execution reaches such a task, then the user is needed to fill the form. Using forms, we can create and update variables that can be used in other tasks and can be used anywhere in the process to control the flow of execution. Each task can be shared with any number of groups and can be assigned to one or more users. An optional due date can also be provided for a task.

Graphical notation



Example


opentext | Workflow Service
Collapsed Sub process
Event sub process
Call activity
Gateways
Boundary Events
Intermediate Catching Ev...
Intermediate timer catching ev...
Intermediate message catchin...
Intermediate signal catching e...
Intermediate Throwing Ev...
Intermediate none throwing ev...
Intermediate signal throwing e...
Intermediate escalation throwi...
Process Navigator
Book travel ticket

Process: StartSignalEventBPM

No structural elements used.

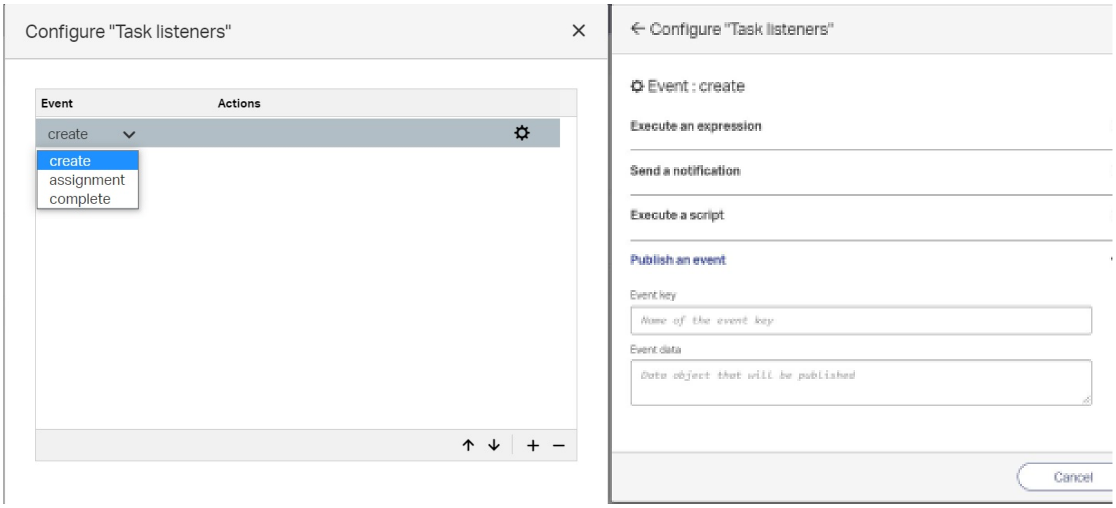
Book travel ticket

Id	No value
Name	Book travel ticket
Asynchronous	□
Exclusive	false
Is for compensation	□
Task nature	No task nature selected
Delivery options	Not configured
Form key	No value
Validate form fields	true
Due date	No value
Priority	No value
Form properties	No form properties selected
Skip expression	No value
Execution listeners	No execution listeners configured
Task listeners	No task listeners configured

Attributes

Group	Attribute	Description
General	ID	Unique identifier of the element within the process model.
	Name	Name of the element. This is the name displayed in the diagram.

Details	Form Properties	Sets the form properties.
	Form Key	Form related URLs to use in applications.
	Validate form fields (server-side)	If validate form fields expression evaluates to true, once the form is submitted then fields of the form are validated as per form

	Task listeners	Used to set task listeners on this task. which allows us to respond to below events: • Create: It happens when task is created and when all properties of task are set. • Assignment: This assignment event is triggered before the Create event, It happens when the task is assigned to someo • Complete: It takes place before the task is removed from the runtime data and when the task is complete. For each event, below actions can be configured. • Execute an expression Given expression is evaluated based on the task instance data. • Send a notification A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is ser Task activity. • Execute a script A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It wo • Publish an event Configure this option to publish an event to OT2 event service for a specific event type. 'Event key' is the event type Io expected in JSON format. Application data can be published to event server as described in Workflow Listeners data Integrati  Configure "Task listeners" Event Actions create create assignment complete ← Configure "Task listeners" Event : create Execute an expression Send a notification Execute a script Publish an event Event key Name of the event key Event data Data object that will be published Cancel
	Asynchronous	When task is set in asynchronous mode, It introduces a wait state when execution reaches to that task. **Note**: In many scenarios, there must not be any reason using this attribute in user task.
	Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same pro
	Execution listeners	Active execution listeners will respond to the process instance events occurred on a Activity. Check Execution Listeners Confi
	Priority	An integer number denoting the priority of the task.
	Due date	Due date of the task. No due date is the default and there are three different options to set a due date: • Relative: Relative to the current date we could add or subtract no. of days, months, or years. • Absolute: An absolute date will be set. • Expressions: An expression is used to calculate the date dynamically.
	Skip expression	Task execution is skipped, If skip expression evaluates to true. **Note**: For skip expression to evaluate there should be a boolean process variable `_WORKFLOW_SKIP_EXPRESSION_ENABLE`
	Is for compensation	If an activity serves as compensation for another activity.
Task Nature	4 eye	4 eye task nature configure states that no two linked task have the same user to work on it. configuring 4 eye is only valid to p Ex: start --->usertask1------>usertask2-----end usertask2 can configure 4 eye nature on usertask1 and vice versa will be ignored.

<table>
<tr><td></td><td>Rendezvous</td><td>Rendezvous task nature configure states that two linked task must have the same user to work on it. configuring Rendezvous

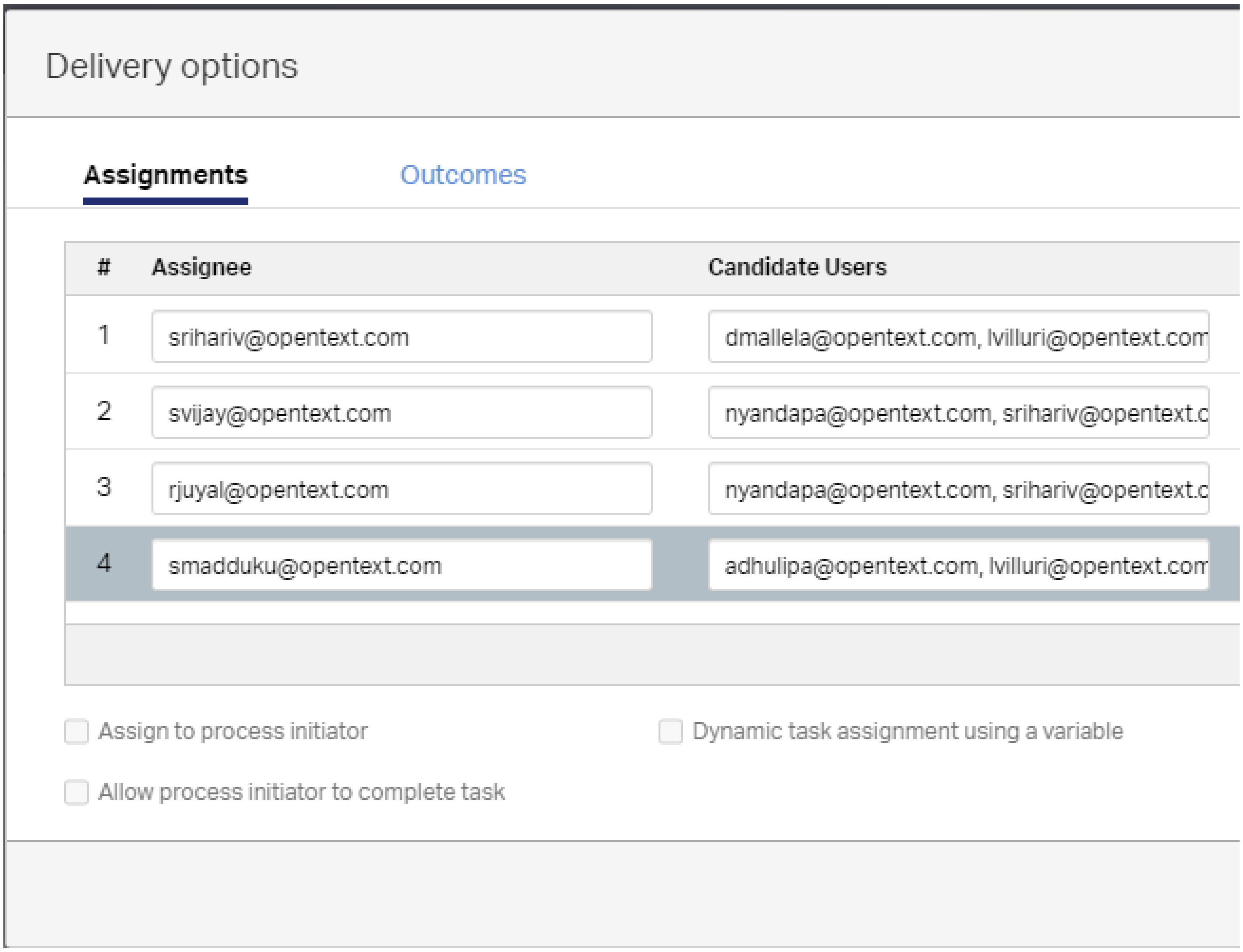
Ex: start --->usertask1------>usertask2-----end

usertask2 can configure Rendezvous nature on usertask1 and vice versa will be ignored.

<b>Note:</b> Task nature properties are missing after importing a model. A open ticket is present (FLOW-986 - Getting issue detai</td></tr>
<tr><td colspan="3"><b>Delivery options</b></td></tr>
<tr><td>Assignments</td><td colspan="2">This tab lets you define one or more assignments on a user task. Configuring a single assignment delivers a single task instance, whereas, configur assignment.

A multi-instance task can be further classified into two categories,
<b>Serial multi-instance task:</b> The multiple instances of the task are delivered one after another to each assignment. Initially, one task instance will be delivered to the second assignment and so on. The order will be the same as the order of the assignments in the table. Uncheck <i>Deliver task to the</i>
<b>Parallel multi-instance task:</b> The instances of the task are delivered to all the configured assignments at once in parallel. Check <i>Deliver task to the</i>

Hereafter, a task with single assignment is referred to as a single-instance task, whereas a task with multiple assignments as a multi-instance task.

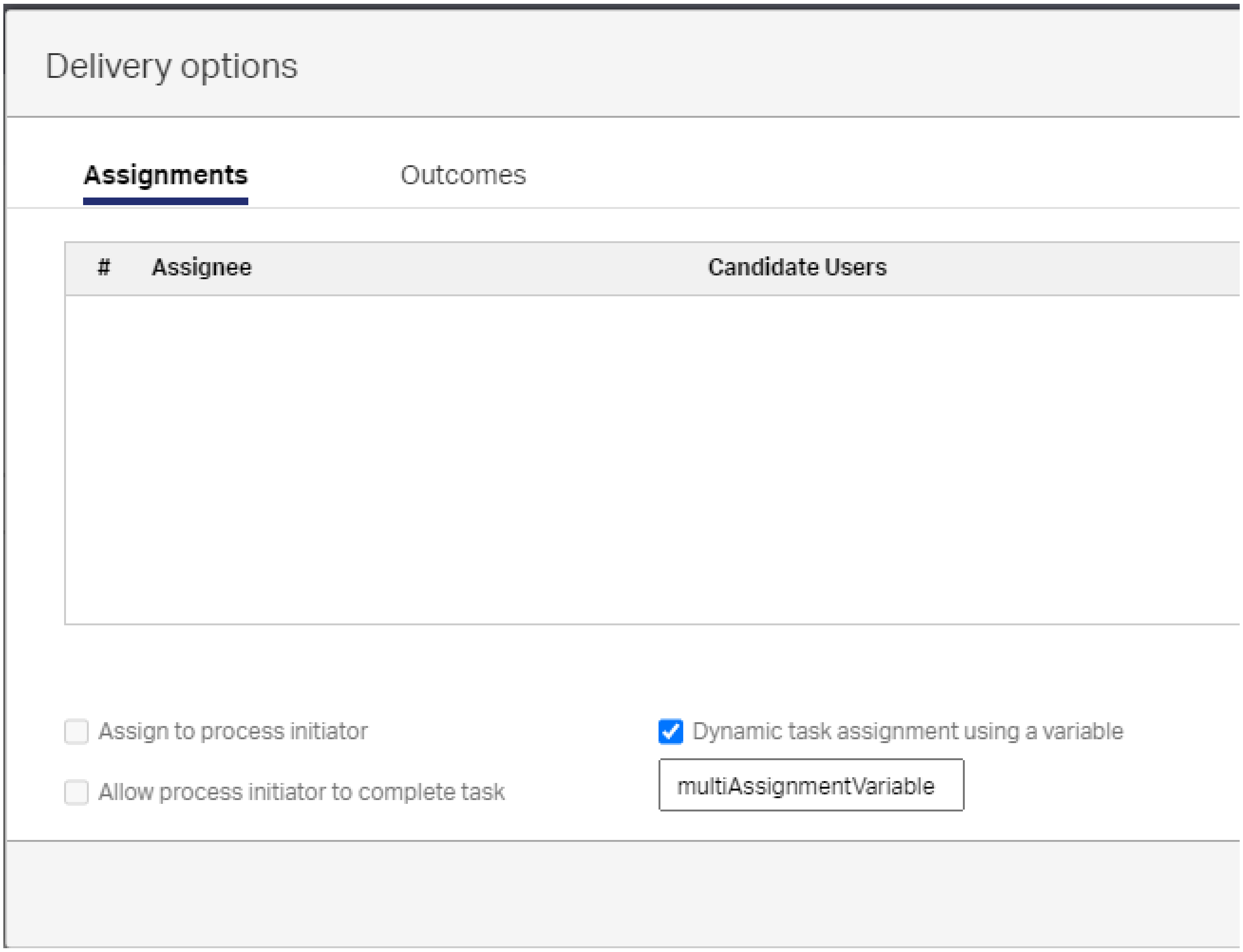

Delivery options

Assignments Outcomes

<table>
<tr><th>#</th><th>Assignee</th><th>Candidate Users</th></tr>
<tr><td>1</td><td>srihariv@opentext.com</td><td>dmallela@opentext.com, lvilluri@opentext.com</td></tr>
<tr><td>2</td><td>svijay@opentext.com</td><td>nyandapa@opentext.com, srihariv@opentext.c</td></tr>
<tr><td>3</td><td>rjuyal@opentext.com</td><td>nyandapa@opentext.com, srihariv@opentext.c</td></tr>
<tr><td>4</td><td>smadduku@opentext.com</td><td>adhulipa@opentext.com, lvilluri@opentext.com</td></tr>
</table>
□ Assign to process initiator □ Dynamic task assignment using a variable
□ Allow process initiator to complete task

<b>Multi-instance task with dynamic assignments:</b> A multi-instance task can be designed to take the assignments at runtime as opposed to pre-defi</td></tr>
</table>

This way you can give a JSON collection variable name through which you intend to insert assignments in the runtime.



Delivery options

#	Assignee	Candidate Users

□ Assign to process initiator

☑ Dynamic task assignment using a variable

□ Allow process initiator to complete task

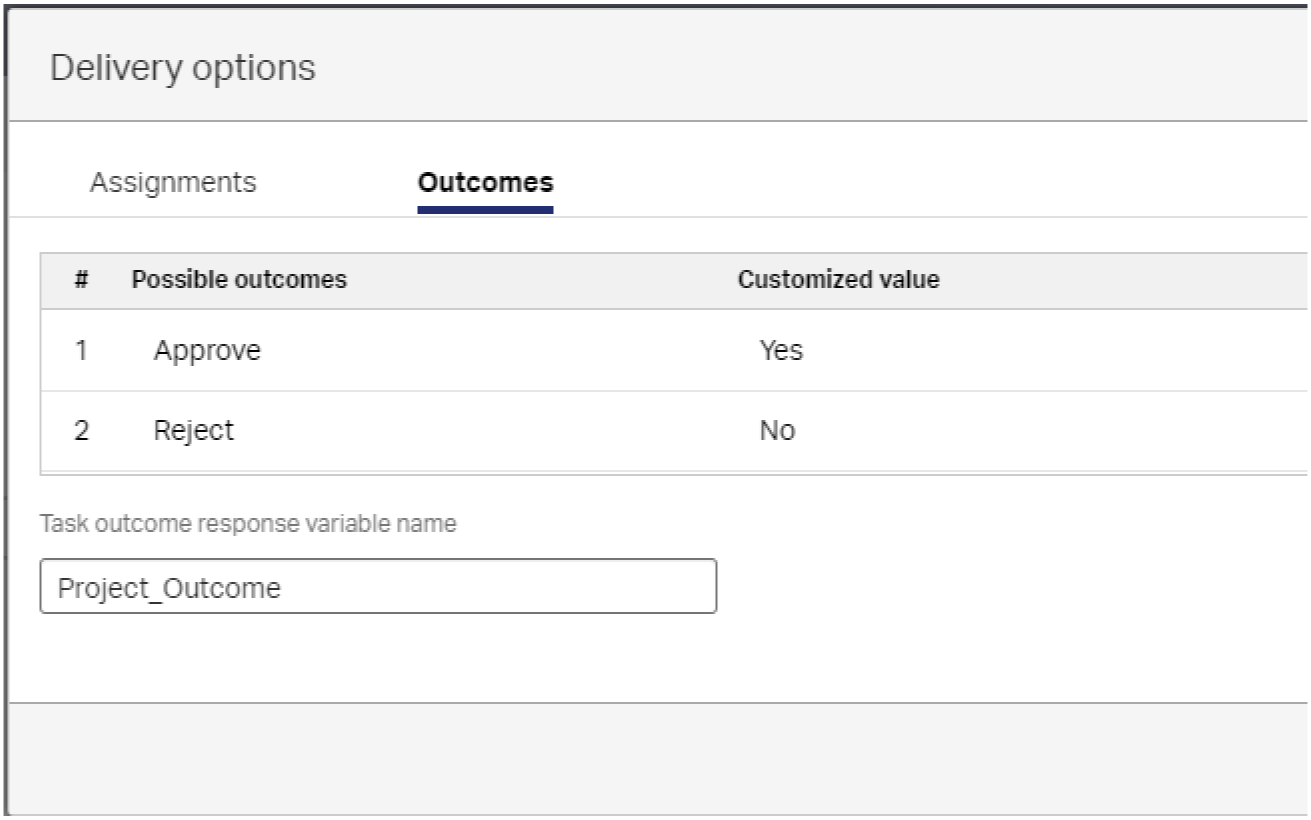
multiAssignmentVariable

The value of the multi-instance dynamic assignment collection variable must be given in the following JSON format:

```
[
        {
                "assignee":"srihariv@opentext.com",
                "candidateUsers":"dmallela@opentext.com,lvilluri@opentext.com",
                "candidateGroups":"Architecture"
        },
        {
                "assignee":"svijay@opentext.com",
                "candidateUsers":"nyandapa@opentext.com,srihariv@opentext",
                "candidateGroups":"Design"
        }
]
```

Task Type	- Default - A normal task with/without a custom set of possible outcomes - Approval - A task with two predefined outcomes *Approve* and *Reject*.
Assignee	Assignee of a task who is responsible for task completion. By default, the assignee is set to `$INITIATOR`, which is a special through an expression. The assignee value should be set to the user's OTDS id or email. The assignee user should be a valid user in the tenant and subscription context (created and managed using Admin Center o
Candidate users	List of users who can become assignees by claiming the task. Update the candidate users directly or use an expression. The candidate users can also be provided. The candidate users should be valid users in the tenant and subscription context (created and managed using Admin Center

	Candidate groups	One or more groups that can become assignees by claiming the task. Update the group directly or use an expression. The ca name, created and managed using Admin Center or ETS. Multiple candidate groups can also be provided. The group should be a valid application role or a group with associated user mappings in the provided tenant and subscription process instance would be created and the task delivered.
	Assign to process initiator	Denotes if the task instance should be assigned to the user who started the process. If enabled, no assignment can be configu Available only on a single-instance task.
	Allow process initiator to complete task	Denotes if the user who started the process is allowed to complete the task. Available only on a single-instance task.
	Deliver task to the assignments in parallel	Checking this option lets the task behave as a parallel multi-instance task, else, as a serial multi-instance task. Available only on a multi-instance task.
Outcomes	This tab lets you configure a set of possible outcomes on the task. The assignee should select one of these possible outcomes on completion of a ta	



Delivery options

#	Possible outcomes	Customized value
1	Approve	Yes
2	Reject	No

Task outcome response variable name: Project_Outcome

	Possible outcomes	The set of possible outcomes from which the assignee can select one as the outcome. • For a Default task, any number of possible outcomes can be configured. • For an Approval task, The possible outcomes are restricted to two predefined values, *Approve* and *Reject*. However, eac *ove* and "No" for *Reject*.
	Customized value	The custom string representation of the outcome. For instance, "Yes" for *Approve* and "No" for *Reject*. Available only on an approval task.
	Condition for completion	On a multi-instance task, the percentage of an outcome needed for completing the task and moving forward to the next activit Eg. Consider a multi-instance approval task with 4 assignments. If the task has to move forward if atleast 2 of the assignments Available only on a multi-instance task.
	Task outcome response variable name	The name of the variable to which the outcome of the task is stored to. This variable can later be used in the flow of the bpm. • For a single-instance task, the outcome variable is stored as a string variable that represents the value of the outcome se value of the selected outcome is stored. Eg. Consider a single-instance task, with outcomes configured as shown in the above screenshot. If the assignee, appro Project_Outcome will be as follows: `"Project_Outcome" : "Yes"`

- For a multi-instance task, the outcome variable is stored as a JSON variable that contains the count of each possible out assignment configured on the user task. The assignments with their specific outcomes are stored in an field *"taskAssignn* following fields:

    - *"assignment"* field will have assignment details(assignee, candidate users and candidate groups) configur
    - *"outcome"* field shows the outcome selected by the assignee. This field is only visible for assignments whc
    - *"status"* field shows the current status of task with the respective assignment. The possible values for this

        1. 'pending' - This is the initial state where for a configured assignment on the user task the workflc
        2. 'active' - This state indicates that a workflow task is created for the specified assignment.
        3. 'complete' - This state indicates that the workflow task is completed by a user by providing an ou
        4. 'skipped' - This state indicates that the assignment is ignored because the completion condition

  Eg1. Consider a multi-instance approval task with 3 assignments. Let the outcomes are configured as shown in the abov completion condition should move the task forward. In this scenario, the value of the outcome variable, Project_Outcome

```
"Project_Outcome" : {
        "Yes" : 2,
        "No" : 0,
        "completionCondition" : "true",
        "taskAssignments": [
                {
                        "assignment":{
                                "assignee": "dmallela@opentext.com",
                                "candidateUsers": "",
                                "candidateGroups": ""
                        },
                        "outcome": "Yes",
                        "status": "complete"
                },
                {
                        "assignment":{
                                "assignee": "",
                                "candidateUsers": "",
                                "candidateGroups": "QA, Dev"
                        },
                        "outcome": "Yes",
                        "status": "complete"

                },
                {
                        "assignment":{
                                "assignee": "",
                                "candidateUsers": "rjuyal@opentext.com, dmallela@opentex
                                "candidateGroups": ""
                        },
                        "status": "skipped"
                }
        ]
}
```

  Eg2. Consider a multi-instance approval task with 3 assignments with sequential delivery. Let the outcomes are configure and the task is pending at second assignee. In this scenario, the value of the outcome variable, Project_Outcome, will be

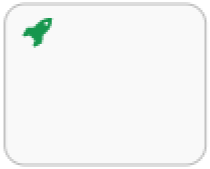
```
"Project_Outcome" : {
        "Yes" : 1,
        "No" : 0,
        "completionCondition" : "false",
        "taskAssignments": [
                {
                        "assignment":{
                                "assignee": "dmallela@opentext.com",
                                "candidateUsers": "",
                                "candidateGroups": ""
                        },
                        "outcome": "Yes",
                        "status": "complete"
                },
                {
                        "assignment":{
                                "assignee": "",
                                "candidateUsers": "",
                                "candidateGroups": "QA, Dev"
                        },
                        "status": "active"

                },
                {
                        "assignment":{
                                "assignee": "",
                                "candidateUsers": "rjuyal@opentext.com, dmallela@opentex
                                "candidateGroups": ""
                        },
                        "status": "pending"
                }
        ]
}
```

⚠ Starting 21.1, Workflow supports OTDS Id for user assignments, retaining the email id support for backward compatibility. Workflow recommends the usage of OTDS Id and plans to deprecate email id support in the coming releases

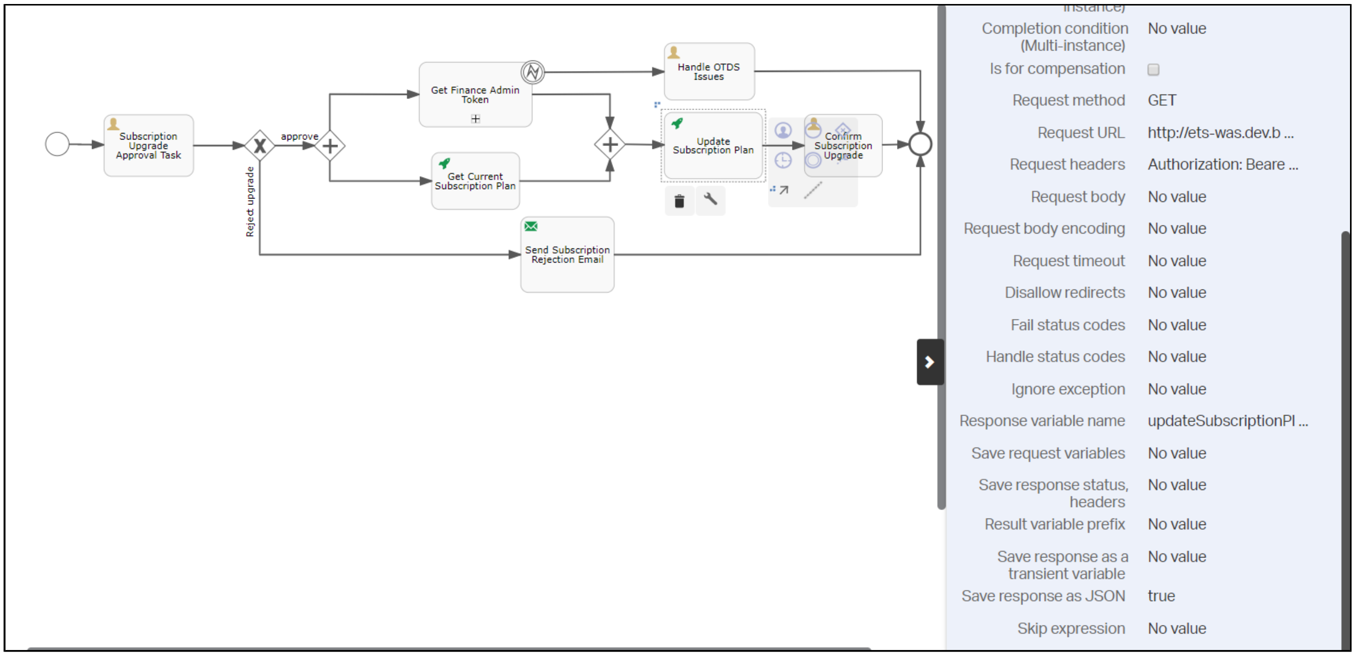
Http task

The HTTP task allows us to make an HTTP call and can store the response.

Graphical notation



Example


Subscription Upgrade Approval Task
approve
Reject upgrade
Get Finance Admin Token
Get Current Subscription Plan
Handle OTDS Issues
Update Subscription Plan
Confirm Subscription Upgrade
Send Subscription Rejection Email

instance)	
Completion condition (Multi-instance)	No value
Is for compensation	□
Request method	GET
Request URL	http://ets-was.dev.b ...
Request headers	Authorization: Beare ...
Request body	No value
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	updateSubscriptionPl ...
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true
Skip expression	No value

Attributes

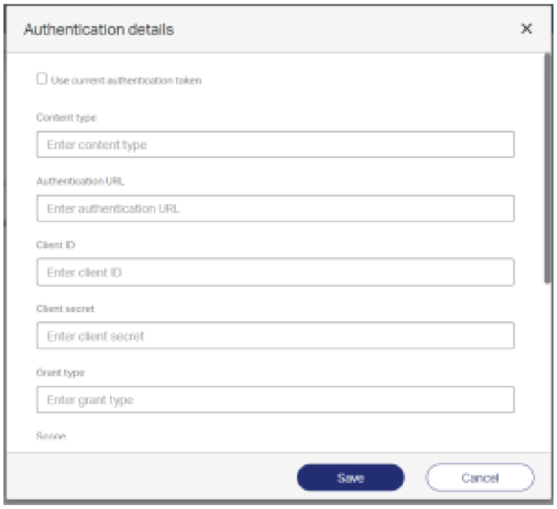
Group	Attribute	Description
General	ID	Unique identifier of the element within the process model.
	Name	Name of the element. This is the name displayed in the diagram.
Details	Request method	Request method to use in the HTTP call: `GET`, `POST`, `PUT`, or `DELETE`.
	Request headers	Line-separated HTTP request headers.
	Request URL	URL of the HTTP request which can contain expressions, for example, http://your-system.example.com/your-endpoint/${someVariable}.
	Request body	Request body to send, for example, a JSON file. Use expressions, for example, `{'clientId': ${clientId}, 'name': ${name}}`.
	Request body encoding	Encoding of the request body.
	Request timeout	Request timeout in milliseconds.
	Disallow redirects	Whether HTTP redirects can be redirected.
	Fail status codes	List of HTTP response status codes to make the request fail and throw a workflow exception. Code ranges can be set with a wildcard `X`, for example, `400`, `404`, and `5XX`.
	Handle status codes	List of status codes for which the task throws a BPMN error, which can be caught by a boundary error event. Code ranges can be set with a wildcard `X`, for example, `400`, `404`, and `5XX`. Status codes in `handleStatusCodes` override those in `failStatusCodes` when they are both set.
	Ignore exception	Whether exceptions are ignored and stored in the variable indicated in the response variable.
	Response variable name	Variable name in which the HTTP response is stored.
	Save request variables	Whether all request variables are stored. By default, only response related variables are stored as variables.
	Save response details	Whether response variables including HTTP status and headers are stored. By default, only the response body is stored as a variable.

Result variable prefix	Prefix is used for easier grouping of result variables, prefix is used before variable names. The affected variables are : `responseStatusCode`, `errorMessage, responseReason, responseProtocol, responseBody and responseHeaders`.
Save response as JSON	Whether the response variable is stored as a JSON variable instead of a string.
Save response as a transient variable	Whether the response variables are stored as transient.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Skip expression	Task execution is skipped, If skip expression evaluates to true. **Note**: For skip expression to evaluate there should be a boolean process variable `_WORKFLOW_SKIP_EXPRESSION_ENABLED` with value `true`**.**
Is for compensation	If an activity serves as compensation for another activity.

<table>
<tr><td></td><td>Authentication Details</td><td>Authentication Details to fetch OAuth2 token for HTTP request.

If 'use current authentication token' selected the access token of current authentication is used for the http request.

Note: This 'use current authentication token' is applicable only for internal OT2 service calls with synchronous execution sequence.



The workflow uses different OTDS authentication grant types such as password, client_credentials nad authorization code to generate the access token.

<table>
<tr><td>Content type</td><td>Which type of data to be sent to OTDS authentication server e.g "application/x-www-form-urlencoded"</td></tr>
<tr><td>Authentication URL</td><td>Provide the OTDS authentication URL</td></tr>
<tr><td>Client ID</td><td>Provide the client identifier provided by the authorization server.</td></tr>
<tr><td>Client secret</td><td>Provide the client secret provided by the authorization server.</td></tr>
<tr><td>Grant type</td><td>Provide the grant type to be used to generate an access token.
Supported grant type to generate an access token for workflow service are password, client_credentials, and authorization_code</td></tr>
<tr><td>Scope</td><td>Provide the scope value given to OTDS to get access token</td></tr>
<tr><td>Client data</td><td>Provide client data value given to OTDS to get access token</td></tr>
<tr><td>Username</td><td>Provide username used with the password grant type</td></tr>
<tr><td>Password</td><td>Provide password used with the password grant type</td></tr>
<tr><td>Redirect URI</td><td>When requesting authorization using the authorization code grant type, specify a redirection URI via the "redirect_uri" parameter.</td></tr>
<tr><td>Code</td><td>Provide an authorization code that the client previously received from the authorization server.</td></tr>
</table></td></tr>
<tr><td>Multi-instance</td><td>Multi instance type</td><td>Whether this activity is performed multiple times and how it is performed. The values are:
• None: The activity is executed once only.
• Parallel : The activity is executed many times with each instance occurring at the same time as others.
• Sequential: The activity is executed many times, one instance following on from the previous one.</td></tr>
<tr><td></td><td>Cardinality (Multi-instance)</td><td>Number of times to perform the activity.</td></tr>
<tr><td></td><td>Collection (Multi-instance)</td><td>Process variable name which contains a collection for each item in this collection, an instance of this activity will be created.</td></tr>
<tr><td></td><td>Element variable (Multi-instance)</td><td>Name of a process variable, which holds the current value of the collection in each activity instance.</td></tr>
</table>

	Completion condition (Multi-instance)	Multi-instance task ends when all instances end. we can provide an expression which can evaluate each time when an instance ends, If that evaluates to `true` all remaining instances will be destroyed and the multi-instance task ends.

Example **request URL**

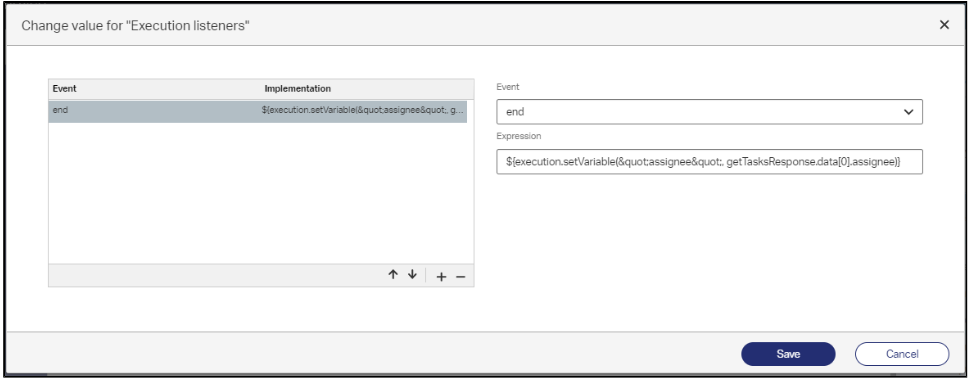
```
https://otdsauth-was.dev.bp-paas.otxlab.net/oauth2/token?
grant_type=password&client_id=pmc&client_secret=qu8l1ty&scope=readwrite
search&username=<<validuser>>&password=<<password>>&client_data=subName%3D<<subscriptionName>>
```

After completion of the HTTP task, the results response is stored in the `Response variable name` in the `JSON/string` format. By default, it is a string. To store the response in the JSON format, set Save response as JSON value to `true`.

Response parsing

As described in Attributes, the response from the HTTP task can be stored as JSON to a variable named `Response variable name` and by setting Save response as JSON to `true`. This can further be parsed and saved to multiple variables using execution listeners. The following steps explain how to perform this with an example:

1. Set `Response variable name` to a name, for example, `getTasksResponse`. Set the Save response as JSON property to `true`.
2. Configure execution listeners on the HTTP service task and create an end event. Write an expression on the event, which obtains a JSONPath value from the `getTasksResponse` variable and sets it to a new variable.



Event	Implementation
end	${execution.setVariable(&quot;assignee&quot;, g...

Event: end

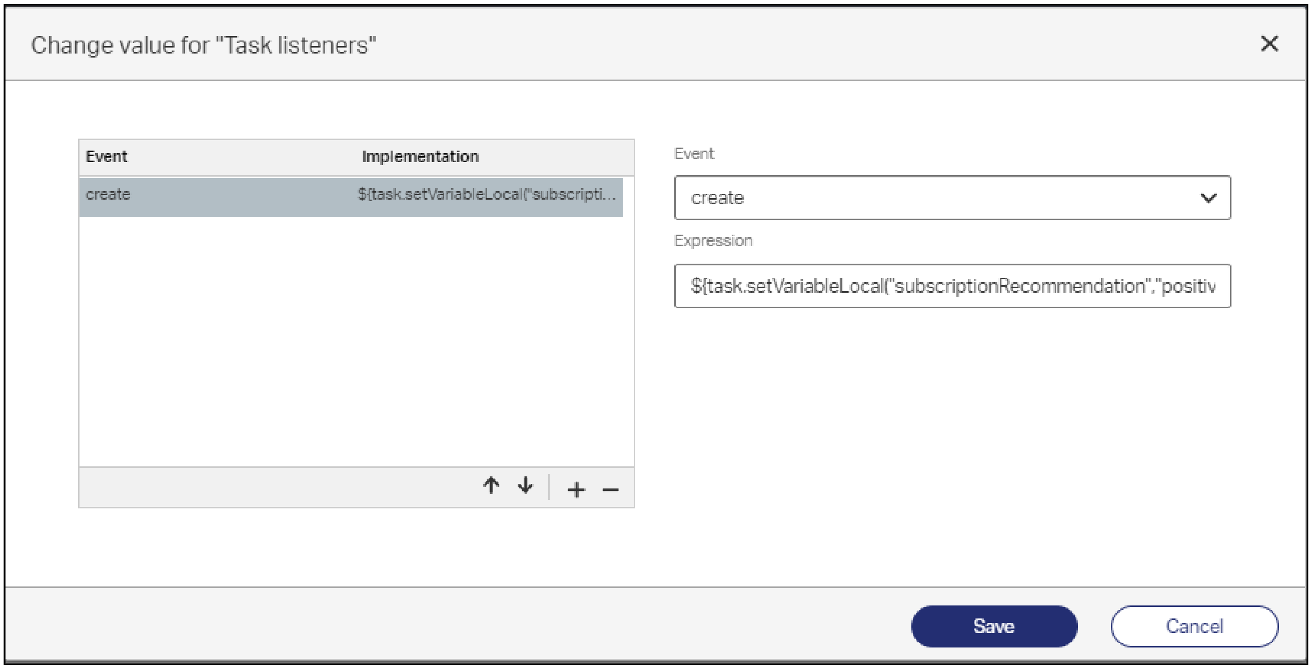
Expression: ${execution.setVariable(&quot;assignee&quot;, getTasksResponse.data[0].assignee)}

The expression in this example is `${execution.setVariable("assignee", getTasksResponse.data[0].assignee)}`, which extracts the assignee from `getTasksResponse` and sets it to a new variable `assignee`.
This adds a new `executionListener` node to the children of the `serviceTask` node in the BPM definition.

```
<workflow:executionListener event="end" expression="${execution.setVariable(&quot;assignee&quot;,
getTasksResponse.data[0].assignee)}"></workflow:executionListener>
```

The execution listeners and task Listeners can be used to create variables for a process instance and task. The variables that are created using task listeners are bound to a task. The following expressions are used to create variables:

- Process variables creation through execution listeners: ${execution.setVariables(variable_name,varaible_value)}
- Task variables creation through task listeners: ${task.setVariableLocal(variable_name,variable_value,true)}
  The last parameter must be true for creating task local variables.



Change value for "Task listeners"

Event	Implementation
create	${task.setVariableLocal("subscripti...

Event: create

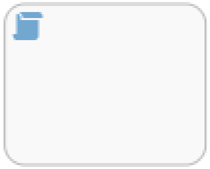
Expression: ${task.setVariableLocal("subscriptionRecommendation","positiv

The supported data types are long, double, string, boolean, and date. The data type is decided based on the the provided value. For example, if the provided value is `Hello`, the string variable is created. If the value is 12, the long variable is created, and if the value is `true` or `false`, the boolean variable is created.

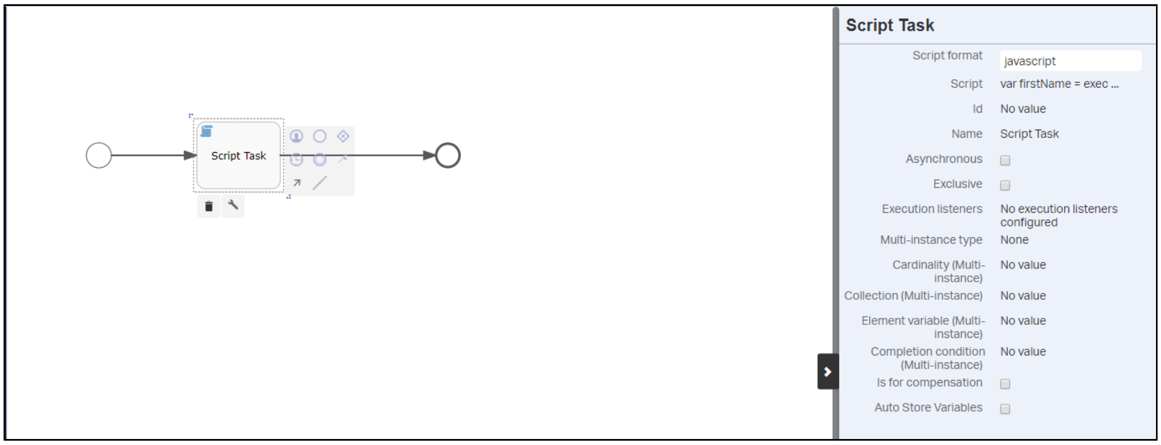
Script task

Business process engine executes a script task. Engine can interpret the language defined by modeler or implementer. Engine executes the script when activity is about to start, When script execution is complete the task also completes. In Workflow, scripts are executed in a JSR-223-compatible scripting language, for example JavaScript. Script tasks are used to perform simple operations or calculations.

Graphical notatio



Example



Script Task

Script format	javascript
Script	var firstName = exec ...
Id	No value
Name	Script Task
Asynchronous	□
Exclusive	□
Execution listeners	No execution listeners configured
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is for compensation	□
Auto Store Variables	□

Example script task (JavaScript) to parse the JSON variable `userInfo` and store in the required variables

```
var userDetails = execution.getVariable("userInfo");
userDetails = JSON.parse(userDetails);
execution.setVariable("firstName", userDetails.firstName);
execution.setVariable("lastName", userDetails.lastName);
```

Attributes

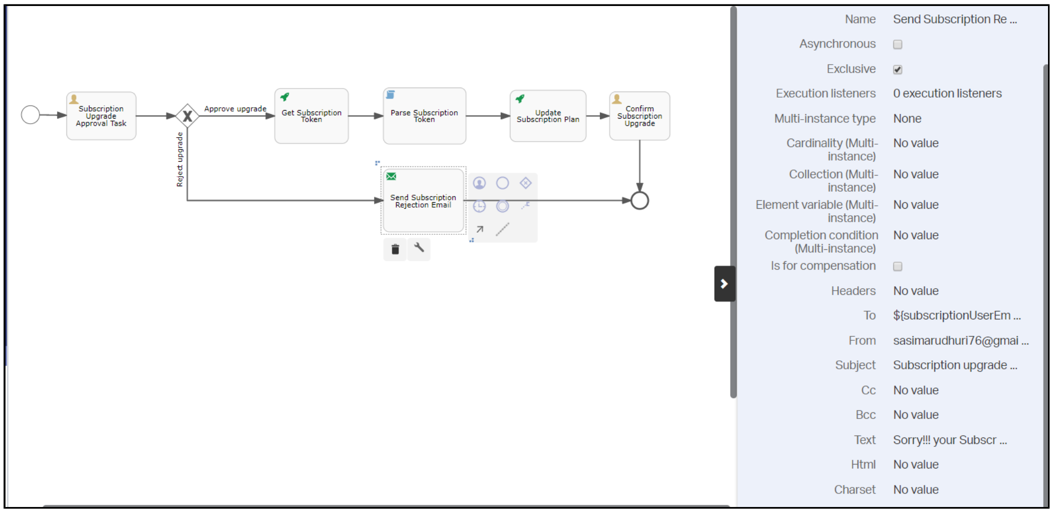
Group	Attribute	Description
General	ID	Unique identifier of the element within the process model.
	Name	Name of the element. This is the name displayed in the diagram.
Details	Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
	Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
	Auto store variables	Automatically stores variables defined in the script during execution. **Note**: Some languages, such as JavaScript do not support this feature.
	Script	Script that executes when the task executes.
	Script format	Format of the script that must be provided if a script is provided. Ex: `JavaScript`.
	Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
	Is for compensation	If an activity serves as compensation for another activity.
Multi-instance	Multi instance type	Whether this activity is performed multiple times and how it is performed. The values are: • None: The activity is executed once only. • Parallel : The activity is executed many times with each instance occurring at the same time as others. • Sequential: The activity is executed many times, one instance following on from the previous one.
	Cardinality (Multi-instance)	Number of times to perform the activity.
	Collection (Multi-instance)	Process variable name which contains a collection for each item in this collection, an instance of this activity will be created.
	Element variable (Multi-instance)	Name of a process variable, which holds the current value of the collection in each activity instance.
	Completion condition (Multi-instance)	Multi-instance task ends when all instances end. we can provide an expression which can evaluate each time when an instance ends, If that evaluates to `true` all remaining instances will be destroyed and the multi-instance task ends.

Mail task

A mail task provides automatic mail services which is used to enhance the business processes, Mail task send emails to multiple recipients. This task support basic email features, like bcc lists, cc lists, and HTML content.

Graphical notation

Example



Name	Send Subscription Re ...
Asynchronous	☐
Exclusive	☑
Execution listeners	0 execution listeners
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is for compensation	☐
Headers	No value
To	${subscriptionUserEm ...
From	sasimarudhuri76@gmai ...
Subject	Subscription upgrade ...
Cc	No value
Bcc	No value
Text	Sorry!!! your Subscr ...
Html	No value
Charset	No value

Attributes

Attribute	Description
Id	Unique identifier for this element.
Name	Name for this element.
To	Receiver email address we can specify multiple Receiver emails by a comma-separated list. Expression can be used when a fixed value is provided. Same as user task, Identity store option is used to select known users or to refer people that were selected in form fields before this email task.
From	Email address of sender. This can be provided as an expression. If this is not given then the default configured system-wide setting `from` address will be used.
Subject	Subject of the email. This can be an expression.
Cc	The CC (Carbon Copy) email recipients. Multiple emails can be added using comma-separated list. This can be provided as an expression.
Bcc	The BCC (Blind Carbon Copy) email recipients. Multiple emails can be added using comma-separated list. This can be provided as an expression.
Text	Email text. If at receiver's end, HTML is not supported then email is displayed in text format (text-only alternative). Specify this and HTML is to support email clients that do not support rich content.
Html	Email content in HTML format. This allows for creation of rich formatting and usage of images.
Charset	Character set for email. By default, UTF8 is used.
Asynchronous	If an activity is made asynchronous then the activity is not executed as part of current action of user, Later this can be helpful if it is not important to have the activity ready immediately.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.

Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: - Start: Happens when the Activity starts. - End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Multi-Instance type	Whether this activity is performed multiple times and how it is performed. The values are: - None: The activity is executed once only. - Parallel : The activity is executed many times with each instance occurring at the same time as others. - Sequential: The activity is executed many times, one instance following on from the previous one.
Cardinality (Multi-instance)	Number of times to perform the activity.
Collection (Multi-instance)	Process variable name which contains a collection for each item in this collection, an instance of this activity will be created.
Element variable (Multi-instance)	Name of a process variable, which holds the current value of the collection in each activity instance.
Completion condition (Multi-instance)	Multi-instance task ends when all instances end. we can provide an expression which can evaluate each time when an instance ends, If that evaluates to `true` all remaining instances will be destroyed and the multi-instance task ends.
Is for compensation	If an activity serves as compensation for another activity.

To configure email in a workflow, add the following configuration in the `workflow-default.properties` file under `C:\Program Files\Tomcat\webapps\workflow-task\WEB-INF - Copy\classes`:

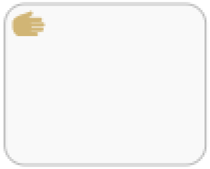
```
workflow.mail.server.host=smtp.gmail.com
workflow.mail.server.port=587
workflow.mail.server.username=******
workflow.mail.server.password=*****
workflow.mail.server.use-tls=true
```

To trigger the mail task, create a process instance. An email is sent to the configured email address.

Manual task

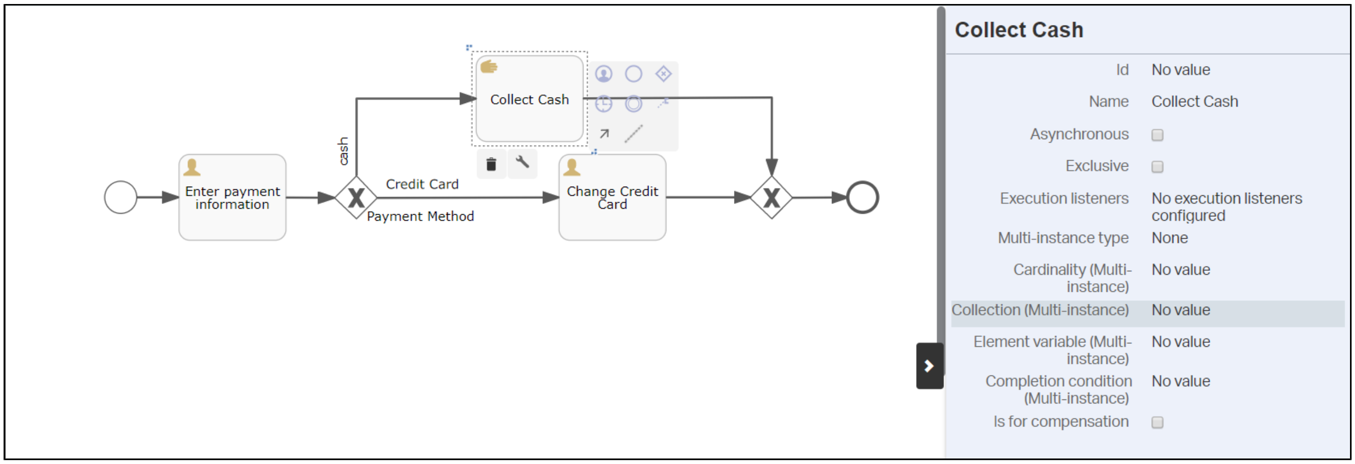
Manual task is external to BPM engine. Business process engine is not aware of work that is performed by this task, Manual task is used to model work that can be performed by someone. Also there is no user interface or system. Manual task is treated as a pass-through task, Where it automatically proceeds with the process from the time execution arrives into it. It is a kind of human task, where some physical interactions are modeled from a business use case and performs them without any involvement of application or business process execution.

Graphical notation



Example

The following example shows how a manual task, which is non-executable (Collect cash) is used to clarify a model.



Collect Cash

Id	No value
Name	Collect Cash
Asynchronous	□
Exclusive	□
Execution listeners	No execution listeners configured
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is for compensation	□

Perform the following steps:

1. Create a process instance in workflow-task.
2. Complete the Enter payment information task by providing the outcome of the form as cash.
3. Complete the Collect Cash task.

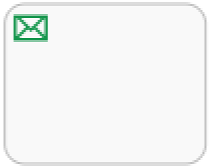
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Multi-Instance type	Whether this activity is performed multiple times and how it is performed. The values are: • None: The activity is executed once only. • Parallel : The activity is executed many times with each instance occurring at the same time as others. • Sequential: The activity is executed many times, one instance following on from the previous one.
Cardinality (Multi-instance)	Number of times to perform the activity.
Collection (Multi-instance)	Process variable name which contains a collection for each item in this collection, an instance of this activity will be created.
Element variable (Multi-instance)	Name of a process variable, which holds the current value of the collection in each activity instance.
Completion condition (Multi-instance)	Multi-instance task ends when all instances end. we can provide an expression which can evaluate each time when an instance ends, If that evaluates to `true` all remaining instances will be destroyed and the multi-instance task ends.
Is for compensation	If an activity serves as compensation for another activity.

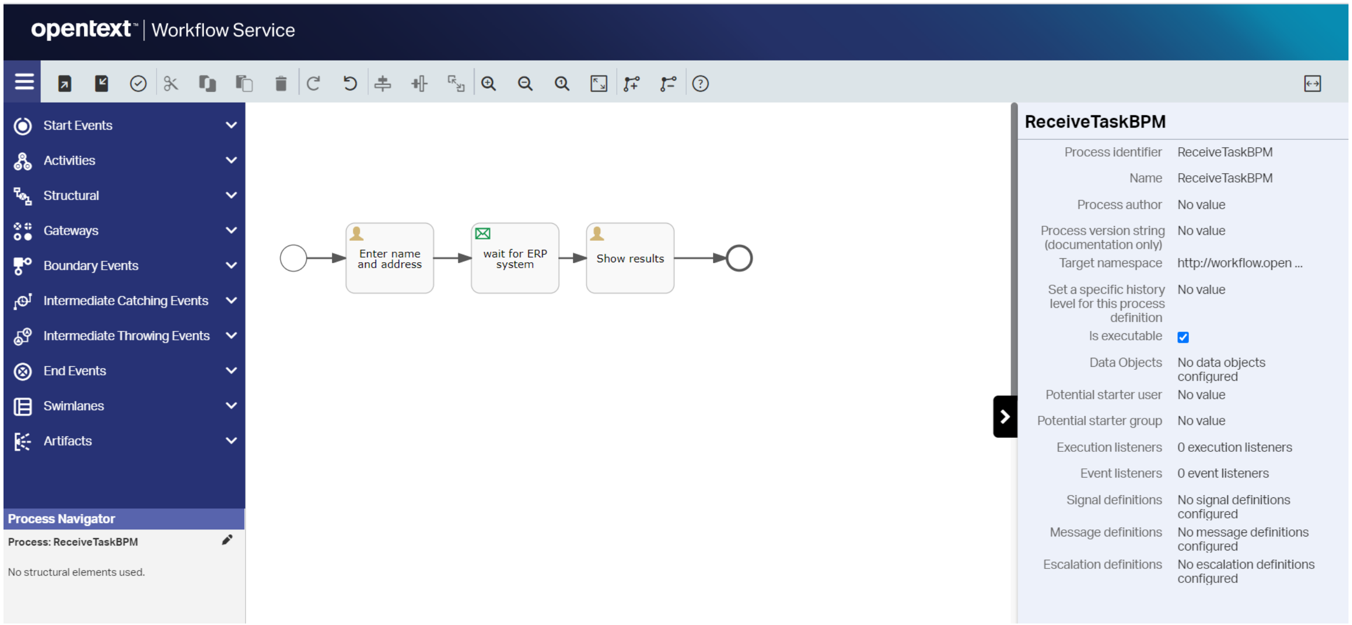
Receive task

A Receive Task waits for a certain message to arrive, When execution arrives at Receive Task, the state of process is committed to persistence storage, i.e. the process will be in waiting state until a specific message is received by the engine, which then triggers continuation of process execution.

Graphical notation



Example



ReceiveTaskBPM

Process identifier	ReceiveTaskBPM
Name	ReceiveTaskBPM
Process author	No value
Process version string (documentation only)	No value
Target namespace	http://workflow.open ...
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	No signal definitions configured
Message definitions	No message definitions configured
Escalation definitions	No escalation definitions configured

Perform the following steps:

1. Create process instance .
2. Complete first user task.
3. Trigger Executions: GET /runtime/executions API to get all executions by giving processInstanceId.
4. Note down value of executionId of receive task.
5. Trigger PUT /runtime/executions/{executionId} by passing executionId of receivetask and action as trigger .
6. Receive Task will be completed and second task will be released.

Attributes

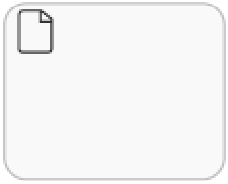
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.

Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Multi-Instance type	Whether this activity is performed multiple times and how it is performed. The values are: • None: The activity is executed once only. • Parallel : The activity is executed many times with each instance occurring at the same time as others. • Sequential: The activity is executed many times, one instance following on from the previous one.
Cardinality (Multi-instance)	Number of times to perform the activity.
Collection (Multi-instance)	Process variable name which contains a collection for each item in this collection, an instance of this activity will be created.
Element variable (Multi-instance)	Name of a process variable, which holds the current value of the collection in each activity instance.
Completion condition (Multi-instance)	Multi-instance task ends when all instances end. we can provide an expression which can evaluate each time when an instance ends, If that evaluates to `true` all remaining instances will be destroyed and the multi-instance task ends.
Is for compensation	If an activity serves as compensation for another activity.

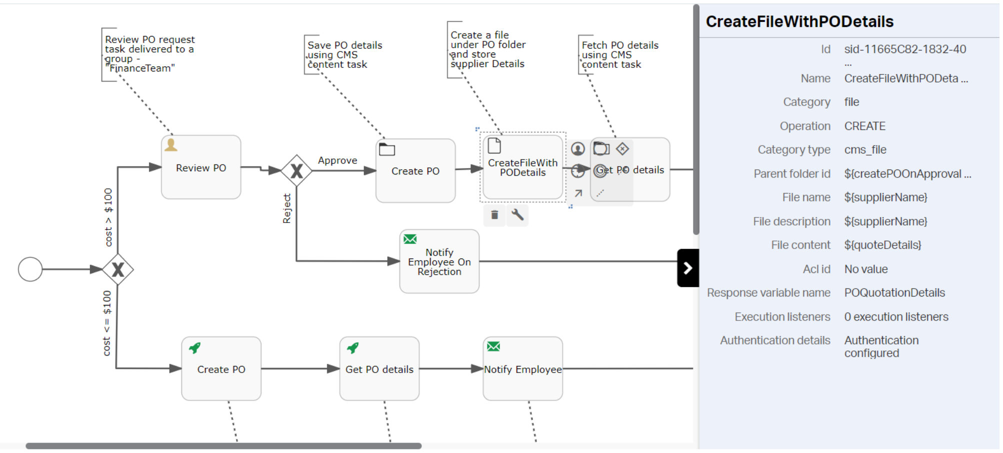
Content task

Content task is used to model the request for content store artifact. Content task gives an abstraction for content store request and stores response in the configured variable. Using content task user can perform CRUD operations on content store object.

Graphical notation



Example



Review PO request task delivered to a group - "FinanceTeam"
Save PO details using CMS content task
Create a file under PO folder and store supplier Details
Fetch PO details using CMS content task
cost > $100
cost <= $100
Review PO
Approve
Reject
Create PO
CreateFileWith PODetails
Get PO details
Notify Employee On Rejection
Create PO
Get PO details
Notify Employee

CreateFileWithPODetails

Id	sid-11665C82-1832-40 ...
Name	CreateFileWithPODeta ...
Category	file
Operation	CREATE
Category type	cms_file
Parent folder id	${createPOOnApproval ...
File name	${supplierName}
File description	${supplierName}
File content	${quoteDetails}
Acl id	No value
Response variable name	POQuotationDetails
Execution listeners	0 execution listeners
Authentication details	Authentication configured

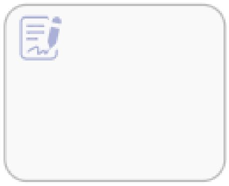
Attributes

Group	Attribute	Description
General	ID	Unique identifier of the element within the process model.
	Name	Name of the element. This is the name displayed in the diagram.
	Authentication Details	Authentication Details configured to fetch OAuth2 token for content store HTTP request
	Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: - Start: Happens when the Activity starts. - End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Content store request details	Category	Content category (As of now limited to File and Folder)
	Operation	Type of the operation required(CRUD)
	Category type	Type definition of the given category(As of now limited to cms_file and cms_folder)
	Response variable name	Process variable to store the response value
	Folder id	Folder identifier in the content store
	Parent folder id	Folder identifier in content store which is used to link it as parent folder
	Folder name	Name of the folder that needs to be created or updated in content store
	Folder description	Description of the folder that needs to be created or updated in content store
	Acl id	Access control list identifier which can be applied to any CMS object to manage the permits of an identity for that object.
	File name	Name of the file that needs to be created or updated in content store
	File description	Description of the file that needs to be created or updated in content store
	File content	Content as static value or from variable

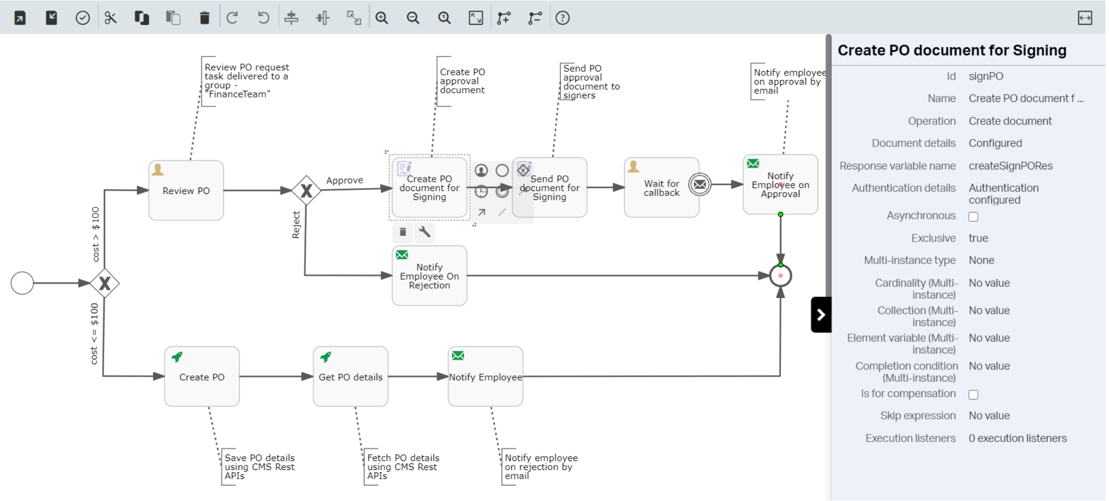
Signature task

Signature task enables users to create documents and send them for signature. Using signature task user can create a document, send a document for signature, delete a document, get documents and signature requests.

Graphical notation



Example


Review PO request task delivered to a group - "FinanceTeam"
Create PO approval document
Send PO approval document to signers
Notify employee on approval by email
Review PO
Approve
Reject
cost > $100
cost <= $100
Create PO document for Signing
Send PO document for Signing
Wait for callback
Notify Employee on Approval
Notify Employee On Rejection
Create PO
Get PO details
Notify Employee
Save PO details using CMS Rest APIs
Fetch PO details using CMS Rest APIs
Notify employee on rejection by email

Create PO document for Signing

Id	signPO
Name	Create PO document f ...
Operation	Create document
Document details	Configured
Response variable name	createSignPORes
Authentication details	Authentication configured
Asynchronous	□
Exclusive	true
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is for compensation	□
Skip expression	No value
Execution listeners	0 execution listeners

Attributes

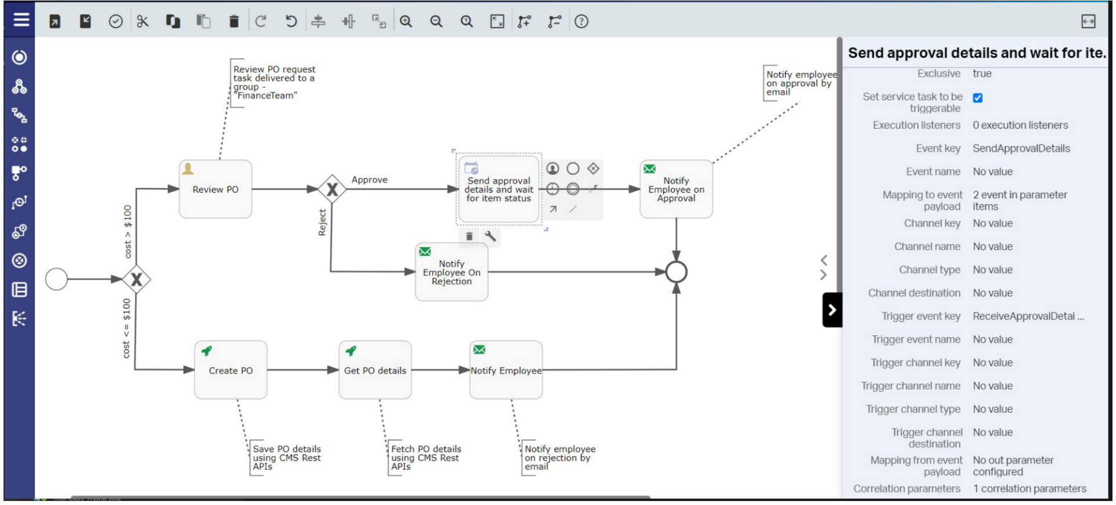
Group	Attribute	Description
General	ID	Unique identifier of the element within the process model.
	Name	Name of the element. This is the name displayed in the diagram.
	Authentication Details	Authentication Details configured to fetch OAuth2 token for signature request.
	Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
	Operation	Type of the operation required (create document, send for signature, delete document, get document or get signature).
	Response variable name	Process variable to store the response value.
Document request details	File name	Defaults to filename.
	File name with extension	Filename including extension.
	File content	Base 64 or Plain document content.
	Public URL	Publicly accessible URL of document to be downloaded by OpenText Core Signature.
	Link expire duration	The number of days for which the download links in the Document Signed email will be valid. Afterwards, they will expire and unauthenticated signers will be unable to download the document. ranges from [1 .. 30].
	Auto expire duration	Number of days after which a non finished document will be automatically expired. ranges from [1 .. 730].
	Auto delete duration	Number of days after which a finished document (signed/cancelled/declined) will be automatically deleted. ranges from [1 .. 730].
	Document URL	URL returned in response when a signature document is created.
	Document ID	UUID returned in response when a signature document is created.
	Force delete	If the document has an unfinished signature request, it will be cancelled. Then, the document will be deleted.
Signature request	First name	First name of signer.

details	Last name	Last name of signer.
	Email	Email of signer.
	Redirect URL	URL where signer needs to be redirected after document has been finished.
	Password	Password which signer needs to enter while performing any action on the document
	Needs to sign	The signer needs to action the document. selected by default.
	Approve only	The signer only needs to approve the document, not sign it.
	Notify only	The signer will only be notified of updates to the signature request. These users cannot perform any actions on the document.
	Subject	Subject of signer email.
	Message	Message to signer email.
	Signature ID	UUID returned in response when a document is sent for signature.

Send Event Task

The send event task can be used to send events to any event registry. It can also wait to receive events by checking the "Set service task to be triggerable" option similar to Boundary event registry event construct.

Example


Review PO request task delivered to a group - "FinanceTeam"
Review PO
Approve
Reject
cost > $100
cost <= $100
Send approval details and wait for item status
Notify Employee on Approval
Notify employee on approval by email
Notify Employee On Rejection
Create PO
Get PO details
Notify Employee
Save PO details using CMS Rest APIs
Fetch PO details using CMS Rest APIs
Notify employee on rejection by email

Send approval details and wait for ite.

Exclusive	true
Set service task to be triggerable	☑
Execution listeners	0 execution listeners
Event key	SendApprovalDetails
Event name	No value
Mapping to event payload	2 event in parameter items
Channel key	No value
Channel name	No value
Channel type	No value
Channel destination	No value
Trigger event key	ReceiveApprovalDetai ...
Trigger event name	No value
Trigger channel key	No value
Trigger channel name	No value
Trigger channel type	No value
Trigger channel destination	No value
Mapping from event payload	No out parameter configured
Correlation parameters	1 correlation parameters

Attributes

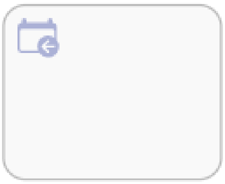
Groups	Attributes	Description
General	ID	Unique identifier of the element within the process model.
	Name	Name of the element. This is the name displayed in the diagram.

	Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Send Event Details	Event Key	Key of the deployed event definition.
	Event name	Name of the event definition.
	Mapping to event payload	The required process variables value can be mapped to the event payload.
Receive Event Details	Set service task to be triggerable	Set this option to receive an event.
	Trigger event key	Key of the deployed event definition.
	Trigger event name	Name of the event definition.
	Correlation parameters	When multiple workflow process instances of the current process definition are running, then the incoming event message can be matched against one of the running workflow process instances.
	Mapping from event payload	The required event payload values can be mapped to process variables.

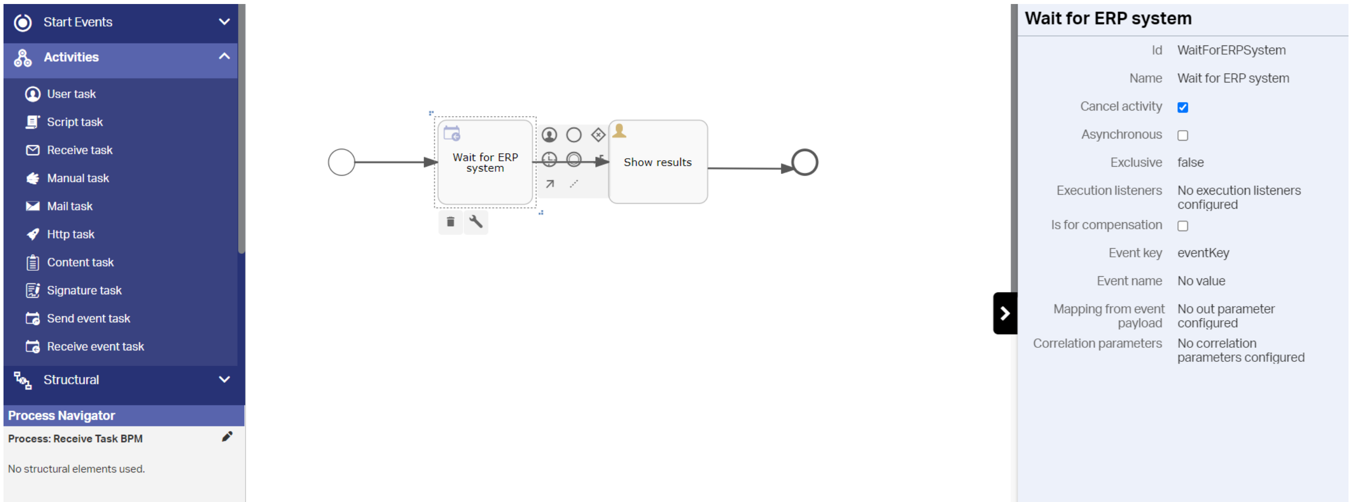
Receive Event Task

Receive event task allows triggering a running process instance with an incoming event, along with correlation. This means when an appropriate event is received through the event registry, the receive event task will be triggered and executed. When this happens, the current activity will be cancelled and the next activity will be created. Please enable asynchronous execution option for subsequent activities in the process to handle and recover any execution errors as deadletter jobs.

Graphical notation



Example



Wait for ERP system

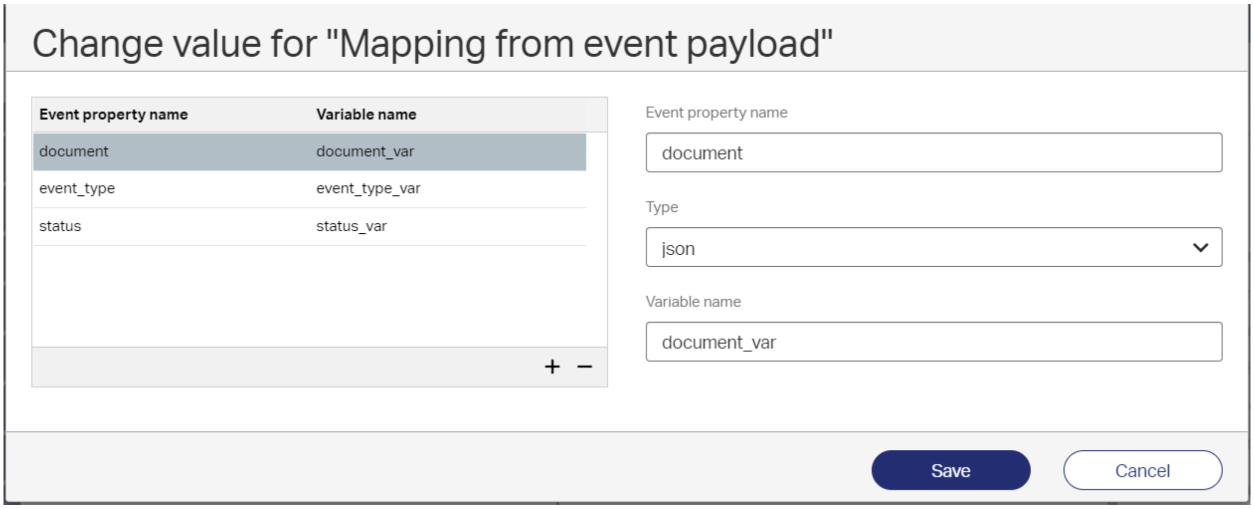
Id	WaitForERPSystem
Name	Wait for ERP system
Cancel activity	☑
Asynchronous	☐
Exclusive	false
Execution listeners	No execution listeners configured
Is for compensation	☐
Event key	eventKey
Event name	No value
Mapping from event payload	No out parameter configured
Correlation parameters	No correlation parameters configured

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners will respond to the following events occurred on a activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Cancel activity	Cancels the activity, if selected.
Event Key	Key of the deployed event definition.
Event name	Name of the event definition.
Mapping from event payload	The required event payload values can be mapped to process variables.
Correlation parameters	When multiple process instances of the current process definition are running, then the incoming event message can be matched against one of the running process instances.
Is for compensation	If an activity serves as compensation for another activity.

Mapping from event payload :

Parameters can be configured to create required process variables from the event payload and can be used anywhere in the execution.



Change value for "Mapping from event payload"

Event property name	Variable name
document	document_var
event_type	event_type_var
status	status_var

Event property name: document

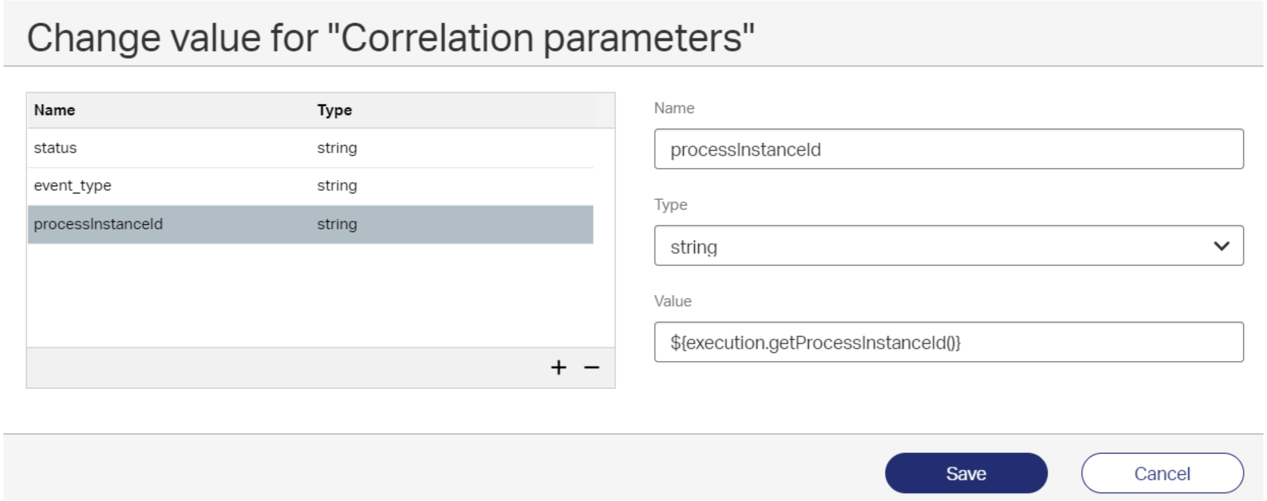
Type: json

Variable name: document_var

In above example three event parameters i.e document, event_type and status from event payload will be mapped to process variables document_var, event_type_var and status_var.

Correlation parameters :

Parameters can be configured to match the values against event payload, we could match multiple correlation parameters, if all of the correlation parameters are matched with the values of the received event payload then only the current activity will be cancelled and moves to the next activities.



Change value for "Correlation parameters"

Name	Type
status	string
event_type	string
processInstanceId	string

Name: processInstanceId

Type: string

Value: ${execution.getProcessInstanceId()}

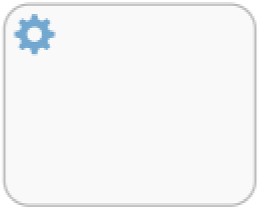
In above example three correlation parameters i.e status, event_type and processInstanceId needs to matched exactly against the status, event_type and processInstanceId of the received event payload.

Here, we are using process instance id as a correlation parameter because when multiple process instances of current process definition are running, then the incoming event payload message can be matched against the current process instance.

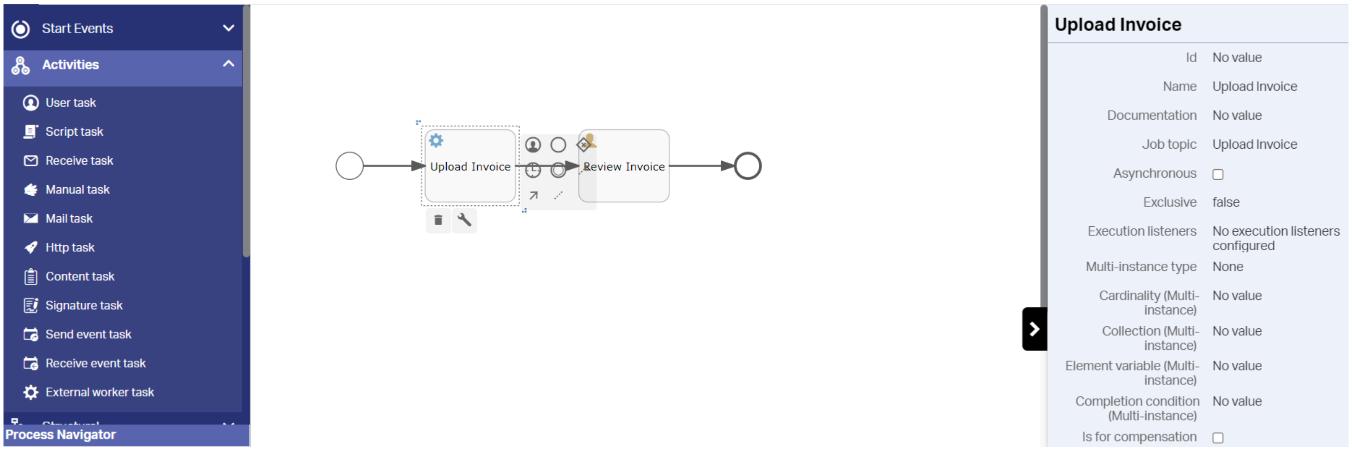
External Worker Task

The External Worker Task allows you to create jobs that should be acquired and executed by External Workers. An External Worker can acquire jobs over the Workflow REST API. An External Worker, which can be implemented in any language, queries Workflow for jobs, executes them, and sends the result to Workflow. The External Worker task is configured by setting the topic which the External Worker uses to acquire the jobs to execute. The external worker can complete or fail the external job, when the retries of the external worker job become 0 then the job will be moved to the failed async job.

Graphical notation



Example



Upload Invoice

Id	No value
Name	Upload Invoice
Documentation	No value
Job topic	Upload Invoice
Asynchronous	□
Exclusive	false
Execution listeners	No execution listeners configured
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is for compensation	□

Attributes

Attribute	Description
Id	Unique identifier of the element within the process model.

Name	Name of the element.
Documentation	Documentation of the element
Job topic	The identifier is used to acquire the external worker jobs to execute
Asynchronous	When this is enabled, the external task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously.
Exclusive	Making a task exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Execution Listeners	Active execution listeners will respond to the process instance events that occurred on an Activity. Check Execution Listeners Configuration
Multi-instance type	Whether this activity is performed multiple times and how it is performed. The values are: • None: The activity is executed once only. • Parallel: The activity is executed many times with each instance occurring at the same time as others. • Sequential: The activity is executed many times, one instance following on from the previous one.
Cardinality (Multi-instance)	The number of times to perform the activity.
Collection (Multi-instance)	Process variable name which contains a collection for each item in this collection, an instance of this activity will be created.
Element variable (Multi-instance)	Name of a process variable, which holds the current value of the collection in each activity instance.
Completion condition (Multi-instance)	The multi-instance task ends when all instances end. we can provide an expression that can evaluate each time when an instance ends, If that evaluates to `true` all remaining instances will be destroyed and the multi-instance task ends.
Is for compensation	If this activity serves as compensation for another activity.

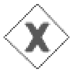
Gateways

Flow of execution is controlled by gateway (or as described by BPMN 2.0 , the tokens of execution). A gateway can generate or consume tokens.

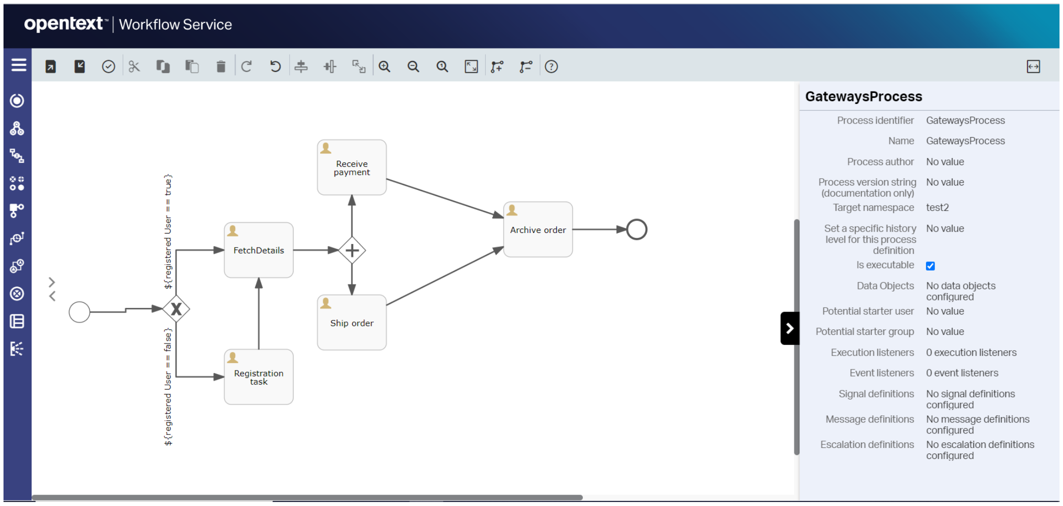
Exclusive Gateway

The XOR gateway is used to model a decision in the process. All outgoing sequence flows will be evaluated when execution arrives at this gateway in the order which they are given. The first sequence flow conceptually whose condition evaluates to true or doesn’t have a condition set on the sequence flow is selected for continuing the process.

Graphical notation



Example


opentext | Workflow Service
${registered User == true}
${registered User == false}
FetchDetails
Registration task
Receive payment
Ship order
Archive order

GatewaysProcess

Process identifier	GatewaysProcess
Name	GatewaysProcess
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	No signal definitions configured
Message definitions	No message definitions configured
Escalation definitions	No escalation definitions configured

Perform the following steps:

1. Create a process instance in workflow-task.
2. Create a variable in workflow-admin and set the variable value to `true/false`. Based on the variable value provided from admin, tasks will be released.

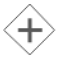
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Flow order	The outgoing flows will be evaluated based on this order and will be stored in the XML representation of process. The outgoing flows order can be managed by clicking on arrows icon.

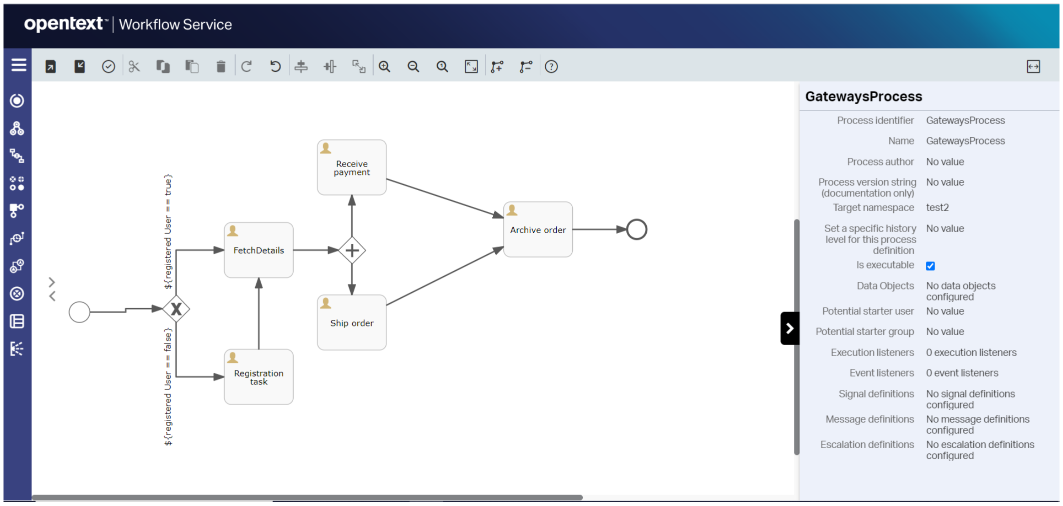
Parallel gateway

Parallel gateway is used to achieve concurrency in a process. This is used to join multiple incoming paths of execution or fork into multiple paths of execution.

Graphical notation



Example


opentext | Workflow Service
Receive payment
FetchDetails
Archive order
Ship order
Registration task
${registered User == true}
${registered User == false}

GatewaysProcess

Process identifier	GatewaysProcess
Name	GatewaysProcess
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	No signal definitions configured
Message definitions	No message definitions configured
Escalation definitions	No escalation definitions configured

After fetch details task, Receive payment and Ship order tasks will be released parallelly.

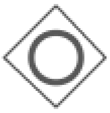
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Flow order	The outgoing flows will be evaluated based on this order and will be stored in the XML representation of process. The outgoing flows order can be managed by clicking on arrows icon.

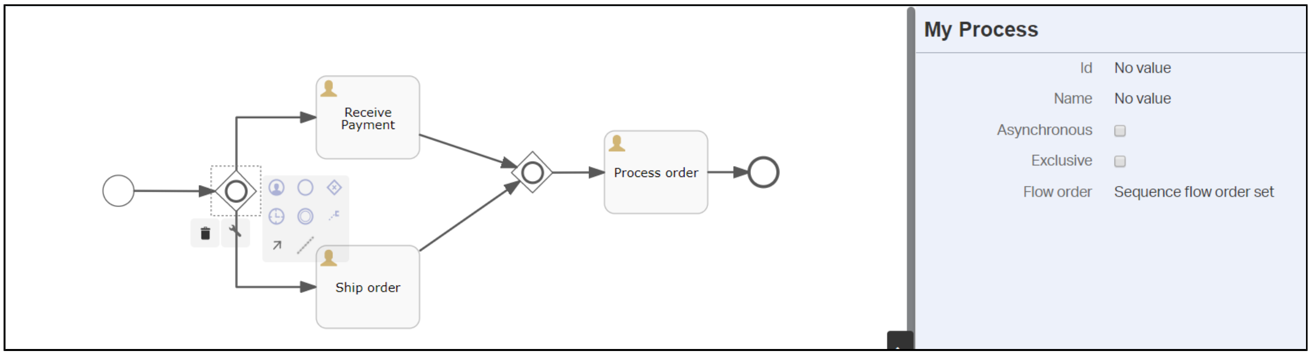
Inclusive gateway

Inclusive gateway is a combination of a parallel and an exclusive gateway. Similar to exclusive gateway, we can provide outgoing sequence flow conditions and it will evaluate them. The difference with exclusive gateway is that inclusive gateway takes more than one path for sequence flow, similar to parallel gateway.

Graphical notation



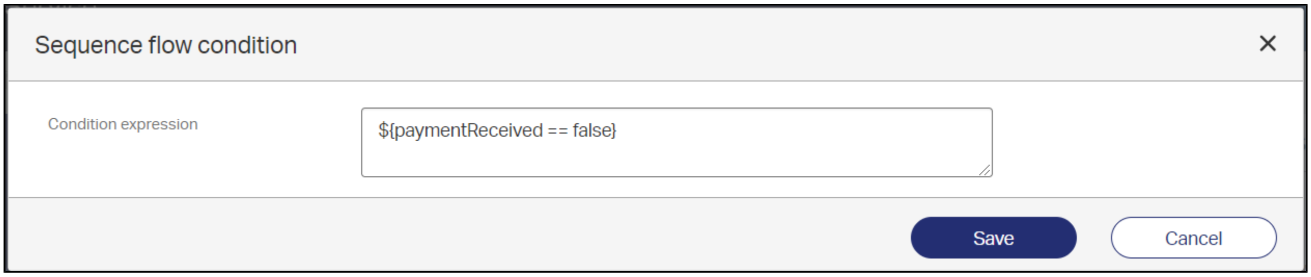
Example



My Process

Id	No value
Name	No value
Asynchronous	□
Exclusive	□
Flow order	Sequence flow order set

Provide the following sequence flow condition:



${paymentReceived == false}

Depending on the condition and variable values, tasks are released.

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Flow order	The outgoing flows will be evaluated based on this order and will be stored in the XML representation of process. The outgoing flows order can be managed by clicking on arrows icon.

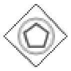
Event based gateway

Event-based gateway takes decisions based on events. Every outgoing sequence flow of event based gateway must be connected to an intermediate catching event, For every outgoing sequence flow an event subscription will be created. When execution reaches event-based gateway, execution will be suspended and it acts as a wait state.

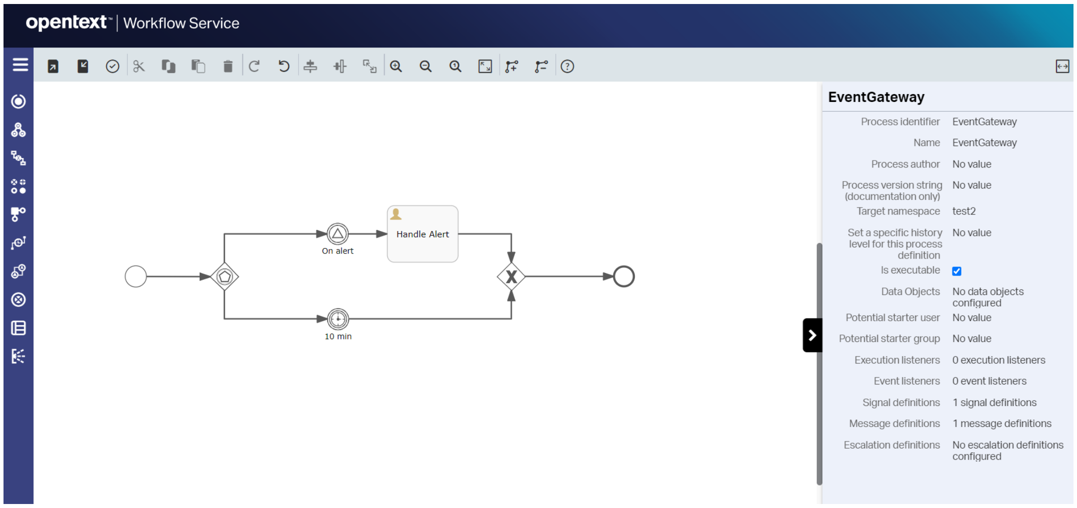
Ordinary sequence flows differ from event-based gateway outgoing sequence flows. These are never actually executed. They allow the engine to determine the events arriving at an event-based gateway that needs to subscribe. The following conditions apply:

- An event-based gateway should definitely have two or more outgoing sequence flows.
- An event-based gateway should only be connected to `intermediateCatchEvent` (Receive tasks are not supported after an event-based gateway).
- An event-based gateway when connected to `intermediateCatchEvent` should have a single incoming sequence flow.

Graphical notation



Example



EventGateway

Process identifier	EventGateway
Name	EventGateway
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	1 signal definitions
Message definitions	1 message definitions
Escalation definitions	No escalation definitions configured

Intermediate timer and signal catching events are configured after the event-based gateway.

Attributes

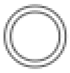
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Exclusive	Making a task as exclusive is useful to solve race conditions, When there are several asynchronous elements of the same process instance then none are executed at the same time.
Flow order	The outgoing flows will be evaluated based on this order and will be stored in the XML representation of process. The outgoing flows order can be managed by clicking on arrows icon.

Intermediate throwing events

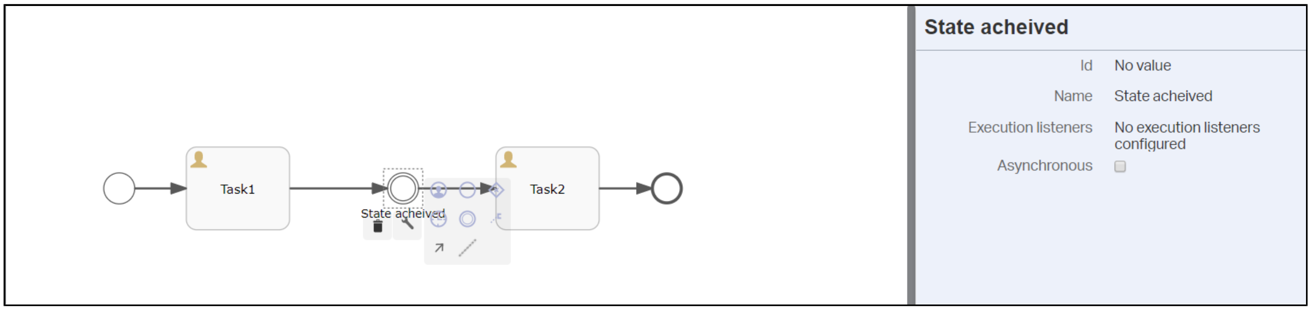
Intermediate throwing none event

Intermediate throwing event is often used to indicate a state achieved in the process. The following model shows a simple example of an intermediate throwing none event.

Graphical notation



Example



State acheived

Id	No value
Name	State acheived
Execution listeners	No execution listeners configured
Asynchronous	☐

The business process engine itself does not do anything in this case; it just passes through.

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.

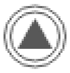
Signal intermediate throwing event

Signal intermediate throwing event throws a signal event for a specified signal.

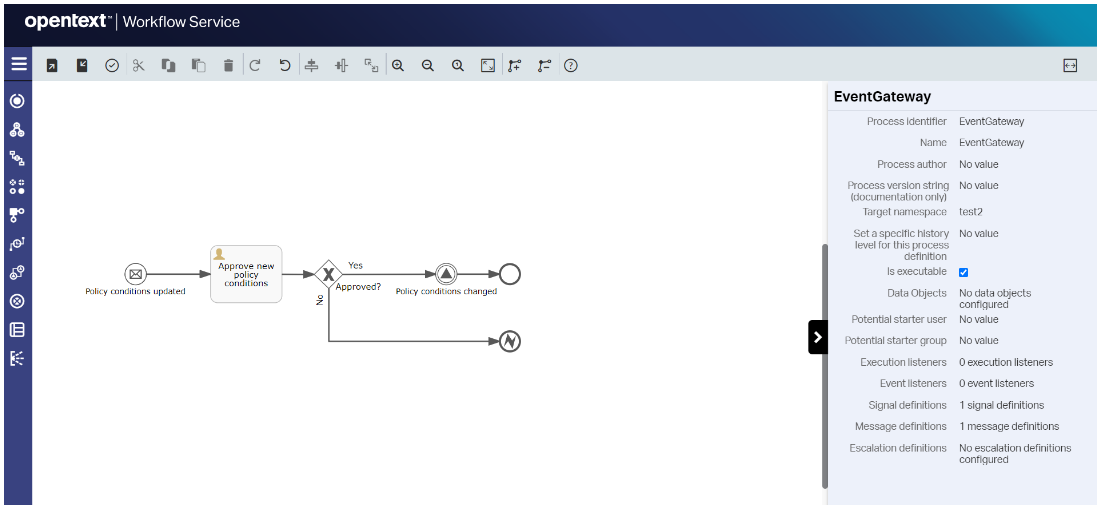
Signal is broadcast to all active catching signal events. Signals can be sent asynchronously or synchronously.

- In default configuration, the signal will be delivered synchronously. The process instance that is throwing signals will wait until all catching process instances receives signal. The signals will be received in same order as the throwing process instance. If one of the received instances produces an error or an exception then all the instances involved will fail.
- A signal can also be delivered asynchronously. In this case ,it is determined which catching signal events are active at the time when throwing signal event is reached. For each active catching signal event, an asynchronous notification message (Job) will be delivered and stored by the JobExecutor.

Graphical notation



Example


opentext | Workflow Service
Policy conditions updated
Approve new policy conditions
Yes
No
Approved?
Policy conditions changed

EventGateway

Process identifier	EventGateway
Name	EventGateway
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	1 signal definitions
Message definitions	1 message definitions
Escalation definitions	No escalation definitions configured

Use the signal thrown in the intermediate throwing event in the defined signal catching event.

Signal catching events will catch the signal thrown by signal throwing event and the configured flow will be started.

Attributes

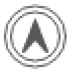
Attribute	Description

ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: - Start: Happens when the Activity starts. - End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Signal reference	Name of the signal.

Intermediate escalation throwing event

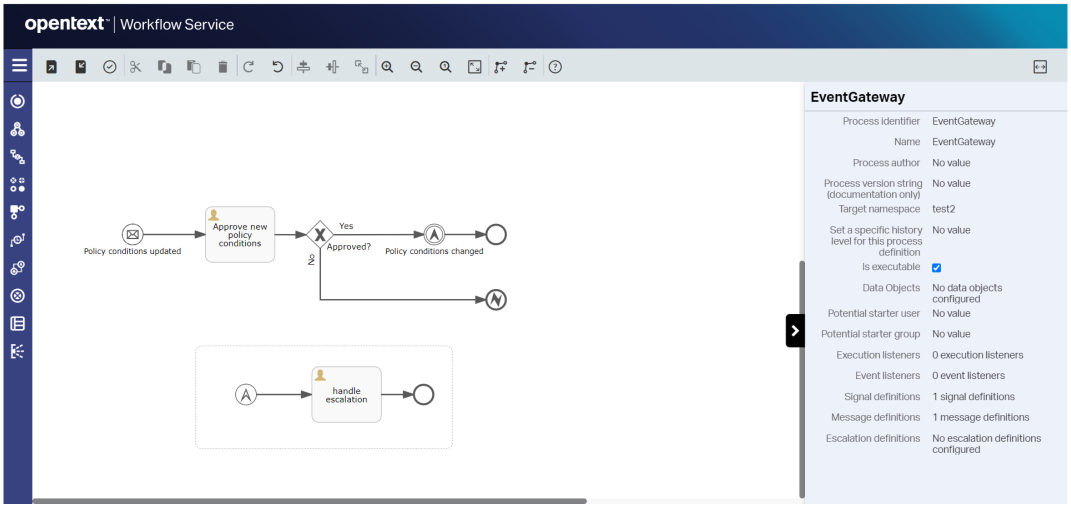
A named escalation will be thrown, when execution reaches at intermediate escalation throwing event. This escalation can be caught by an event sub-process with an escalation start event or an escalation boundary event, which has the none or same escalation code.

Graphical notation



Example

The following example shows how to configure an escalation throwing event and the corresponding escalation start event defined in the event sub-process.



opentext™ | Workflow Service
Policy conditions updated
Approve new policy conditions
Yes
Approved?
No
Policy conditions changed
handle escalation

EventGateway

Process identifier	EventGateway
Name	EventGateway
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	1 signal definitions
Message definitions	1 message definitions
Escalation definitions	No escalation definitions configured

Define escalation definitions in the model and select using Escalation reference while configuring the throwing or start escalation event.

The model execution is as follows:

1. A process instance is created.
2. Complete task 'Approve new policy conditions'.
3. If output of the gateway is yes, intermediate escalation throwing event will be executed.
4. An escalation throwing event throws the escalation. Intermediate escalation catching event defined in the event sub-process, catches the escalation reference and handle escalation task will be released.

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.

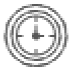
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When this is enabled, the task is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This is used when the task execution takes much time to return to the user interface. but, if any error occurs before the following wait state, there won't be direct user feedback.
Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: • Start: Happens when the Activity starts. • End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Escalation reference	Name of the escalation.

Boundary events

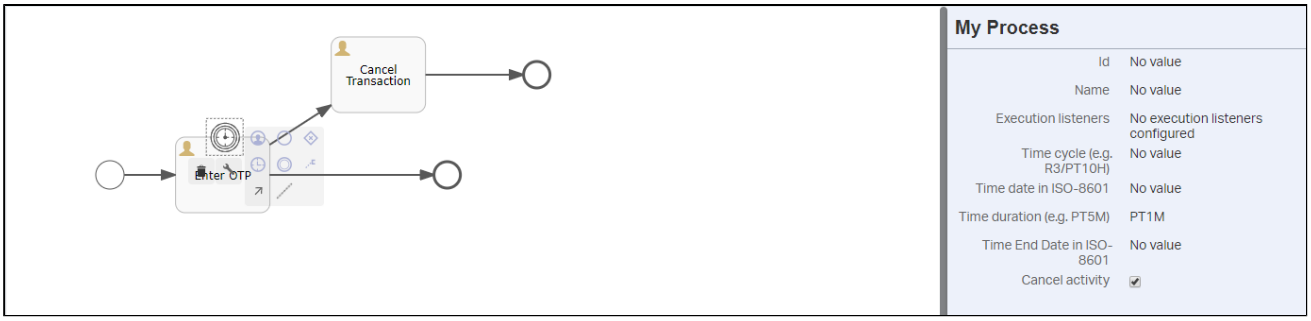
Timer boundary event

Timer boundary event behaves as alarm clock and stopwatch. Once execution reaches at an activity to which this boundary event is attached, a timer starts. The activity is interrupted after a specified interval and the outgoing sequence flow of the boundary event is followed.

Graphical notation



Example



My Process

Id	No value
Name	No value
Execution listeners	No execution listeners configured
Time cycle (e.g. R3/PT10H)	No value
Time date in ISO-8601	No value
Time duration (e.g. PT5M)	PT1M
Time End Date in ISO-8601	No value
Cancel activity	☑

The model execution is as follows:

1. A process instance is created.
2. When an execution arrives at the activity where the boundary event is attached, a timer starts.
3. When the timer fires (for example, after a specified interval), the activity is interrupted and the outgoing sequence flow of the boundary event is followed.

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.

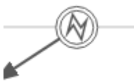
Execution listeners	Active execution listeners will respond to the following events occurred on a Activity: - Start: Happens when the Activity starts. - End : Happens when the Activity completes. A mail notification can be configured to be sent when a specific listener event is triggered. On enabling it, a mail is sent synchronously with the listener event execution. It works similar to the Mail Task activity. A custom script can also be configured to be executed synchronously when a specific listener event is triggered. It works similar to the Script Task activity.
Time cycle (e.g. R3 /PT10H)	Defines a repeating time interval, This is used to send multiple reminders for a delayed user task or to start the process periodically. Time cycle component can be in repeating time duration format as defined by the ISO 8601 standard. example, three repeating time intervals, lasting 10 hours each.
Time date in ISO-8601	Specifies a fixed date (ISO 8601 format) when the trigger will fire.
Time duration (e.g. PT5M)	Specifies how long the timer must run before it is fired. A `timeDuration` can be defined as a sub-component of `timerEventDefinition` . The ISO 8601 format is used as required by the BPMN 2.0 specification.
Cancel activity	Cancels the activity, if selected.

Error boundary event

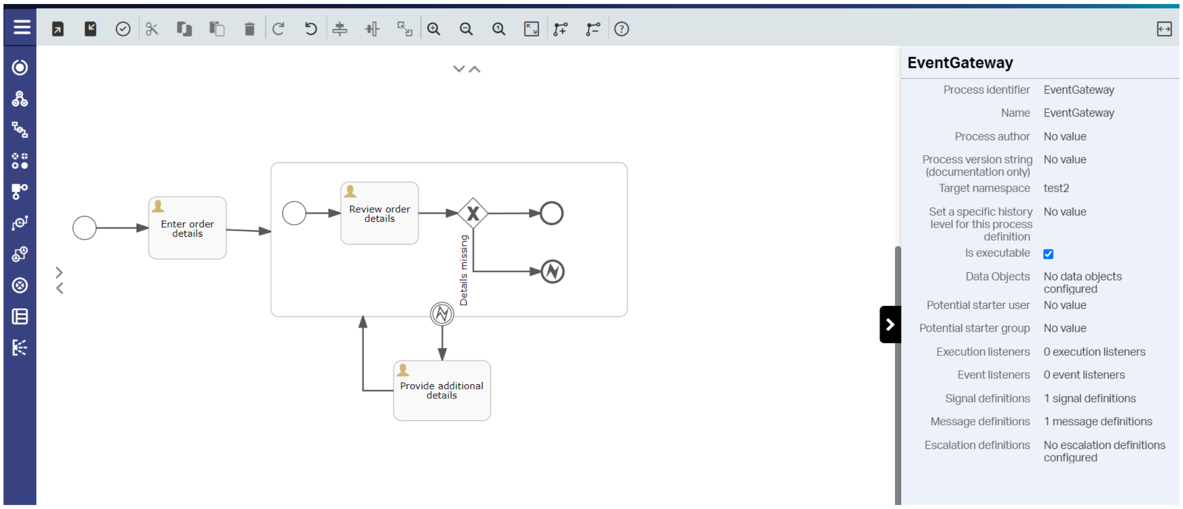
Boundary error event catches the errors that are thrown within the scope of the task to which it is attached. Defining a boundary error event makes more sense on a call activity or an embedded sub-process because a sub-process creates a scope for all tasks inside the sub-process. Error end events will throw these errors. Such an error propagates to its parent scopes upwards until a scope is found on which the defined boundary error event matches the error event definition.

When an error event is caught, the task to which the boundary event is attached will be destroyed, along with all current executions therein (for example, nested sub-processes and concurrent activities). Execution continues through the outgoing sequence flow of the boundary event.

Graphical notation



Example



EventGateway

Process identifier	EventGateway
Name	EventGateway
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	1 signal definitions
Message definitions	1 message definitions
Escalation definitions	No escalation definitions configured

Select the error reference value in both the error boundary event and error end event.

In the error end event, provide the sequence flow condition as `${enoughinformation == false}`.

The model execution is as follows:

- A process instance is created.
- The variable value `detailsMissing` is set to `true`.
- Complete Enter order details task.
- As value of detailsMissing is set to `true`, execution ends and the error is caught.
- The boundary error event catches the error and Provide additional details task will be released.

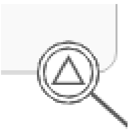
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Error reference	Name of the error.

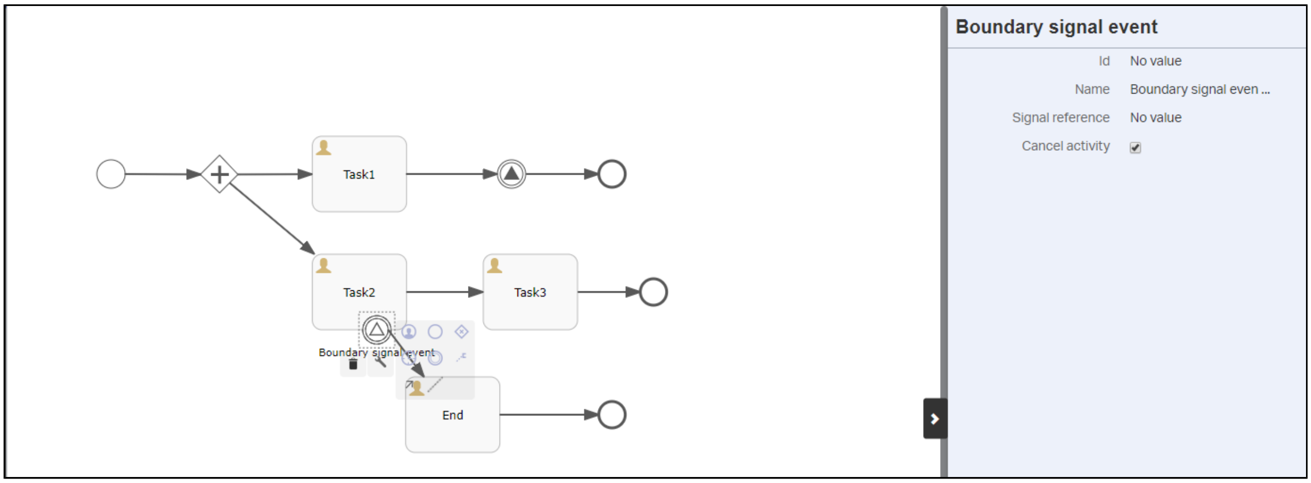
Signal boundary event

When an execution arrives to the task to which the signal boundary event is attached, the signal boundary event catches signals with the proper name. Contrary to other events, such as the error boundary event, a signal boundary event does not only catch signal events thrown from the scope it is attached to. A signal event has a global scope (broadcast semantics), meaning that the signal can be thrown from any place, even from a different process instance.

Graphical notation



The following model shows how to configure the signal boundary event and runtime execution.



Boundary signal event

Id: No value
Name: Boundary signal even ...
Signal reference: No value
Cancel activity: ☑

Create a new signal reference and select in signal throwing and boundary events.

The model execution is as follows:

1. A process instance is created.
2. Task1 completes.
3. An intermediate signal throw event catches the signal and the boundary signal event, which is configured at Task2 catches the signal, and the End task is released.
4. Task2 and Task3 are skipped after the boundary signal event.

Attributes

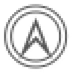
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Signal reference	Name of the signal.

Cancel activity	Cancels the activity, if selected.

Escalation boundary event

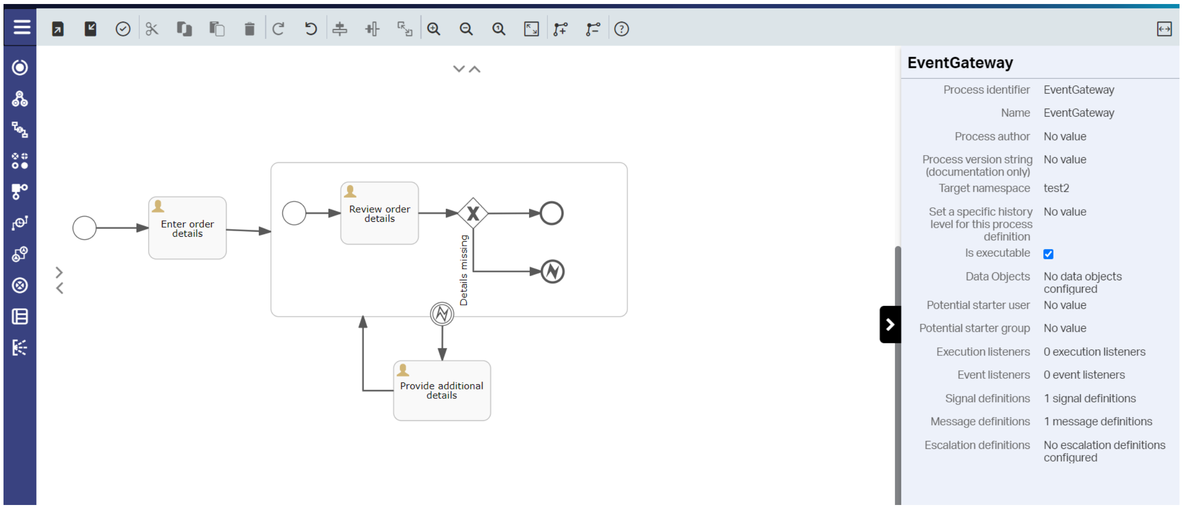
Escalation boundary event catches escalations that are thrown within the scope of the task on which it is defined. It can only be attached to a call activity or an embedded sub-process because only an escalation end event or an escalation intermediate throw event can thrown an escalation. When an escalation event from a call activity triggers the boundary event, the output variables defined on the call activity are passed to the scope of the boundary event.

Graphical notation



Example

The following BPMN model shows how to configure an escalation boundary event.



EventGateway

Process identifier	EventGateway
Name	EventGateway
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	1 signal definitions
Message definitions	1 message definitions
Escalation definitions	No escalation definitions configured

Create the escalation definition and use the value while adding the escalation throwing and boundary escalation events.

The model execution is as follows:

1. A process instance is created.
2. The Enter order details task completes.
3. Complete Review order details task.
4. At the gateway, provide a Boolean variable with the escalation value `true`.
5. The escalation throw event throws an escalation, the escalation boundary event at the sub-process catches the escalation, and the Provide additional details
6. task releases.

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Escalation reference	Name of the escalation.

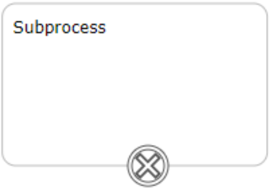
Cancel and compensation boundary events

Cancel event on the boundary of a transaction sub-process is triggered when a transaction is canceled. When it triggers, at first it will interrupts all executions that are active in current scope. Next, it starts compensation of all active compensation boundary events in the scope of the transaction. Compensation is performed synchronously, that means the boundary event waits until compensation is complete before leaving the transaction. When compensation is complete, the transaction sub-process is left through the outgoing sequence flows of the cancel boundary event.

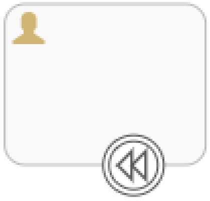
Note: The cancel boundary event must be configured with the cancel end event and the boundary compensation event. User task is not supported for configuring the compensation event. Therefore, you can use the HTTP task while adding a compensation event.

Graphical notation

Boundary cancel event

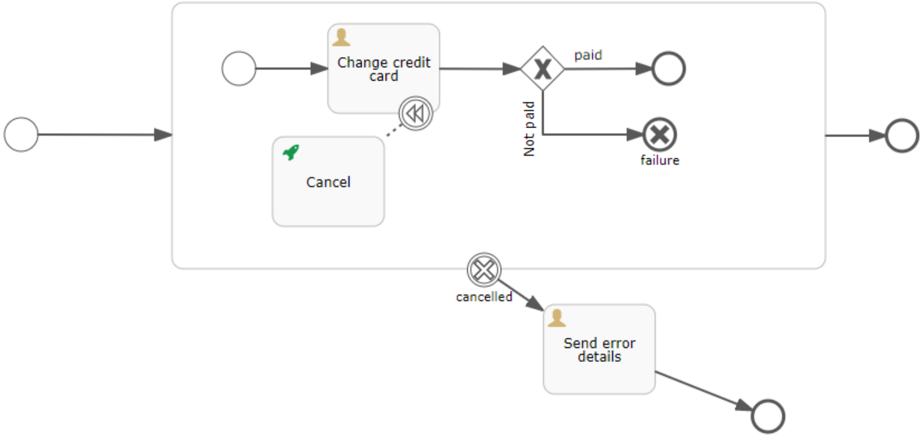


Boundary compensation event



Example

The following example shows how to use cancel and compensation boundary events together in a transaction sub-process.



Following are the configurations.

- Check Is: A transaction sub-process check box.
- Check Is for compensation: A check box for an HTTP task, which is added at the boundary compensation event.

The model execution is as follows:

- A process instance is created.
- The Complete charge credit card task completes.
- Provide a condition at the gateway such that the cancel end event must trigger: `(${paid == false})`
- Compensation occurs after the cancel end event.
- The boundary cancel event executes and the Send error details task releases.

Attributes

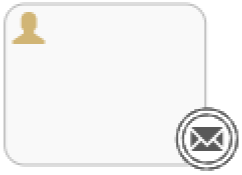
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.

Boundary message event

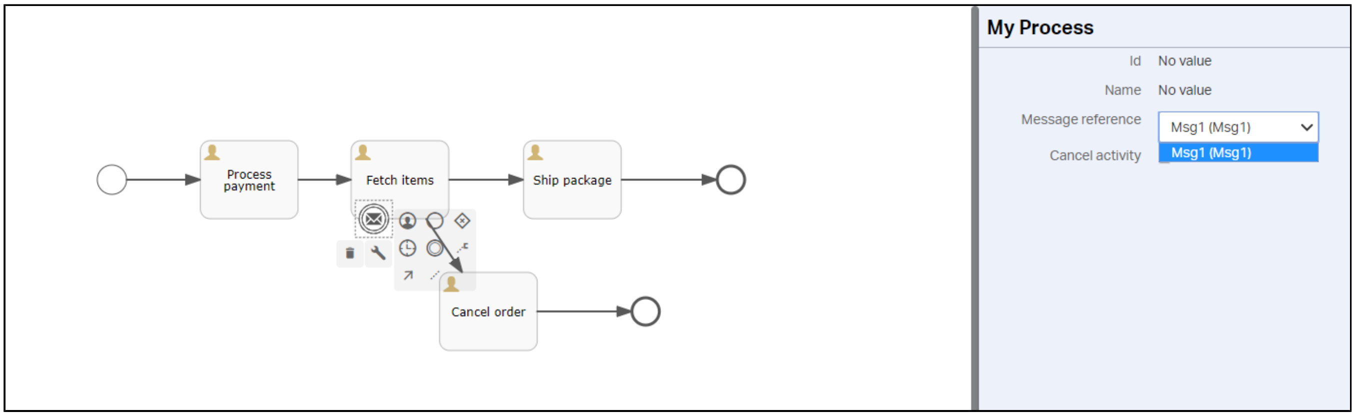
Boundary message event when attached to a task, will be listening for named message. When this named message is caught depending on the configuration of the boundary event following two things can happen:

- Interrupting boundary event: The task is interrupted and the sequence flow going out of the event will be followed.
- Non-interrupting boundary event: One token stays in the task and an additional token is created which follows the sequence flow going out of the event.

Graphical notation



Example



My Process

Id No value

Name No value

Message reference Msg1 (Msg1)

Cancel activity

Msg1 (Msg1)

Perform the following steps:

1. Create process instance in workflow-task.
2. Complete Process payment task.
3. When the execution arrives at Fetch items task, Through swagger note down process instance id.
4. To get the required process instance executionId, use the rest API 'POST /query/executions
5. To trigger the message event on the given process instance use the rest API '/runtime/executions/{executionId}', pass the message name in the request body.
6. On receiving message, Message boundary event will be activated and Cancel order task will be released.

Attributes

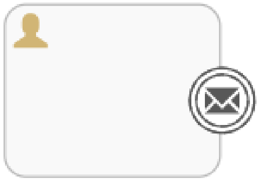
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Message reference	Name of the message.
Cancel activity	Cancels the activity, if selected.

Boundary event registry event

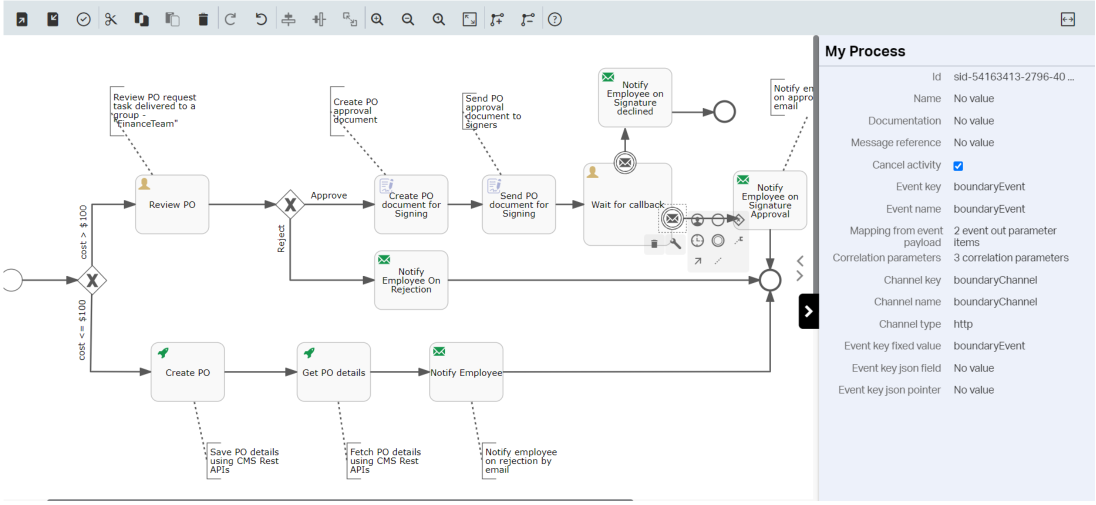
Boundary event registry event allows triggering a running process instance with an incoming event, along with correlation and tenant detection support. This means when an appropriate event is received through event registry, the boundary message will be triggered and executed. When this happens, the current activity will be cancelled and the next activity will be created. Any number of Boundary event registry events can be attached to user and receive activities.

Please enable asynchronous execution option for subsequent activities in the process to handle and recover any execution errors as deadletter jobs.

Graphical notation



Example



Review PO request task delivered to a group - "FinanceTeam"
Create PO approval document
Send PO approval document to signers
Notify Employee on Signature declined
Notify em on appro email
cost > $100
cost <= $100
Review PO
Approve
Reject
Create PO document for Signing
Send PO document for Signing
Wait for callback
Notify Employee on Signature Approval
Notify Employee On Rejection
Create PO
Get PO details
Notify Employee
Save PO details using CMS Rest APIs
Fetch PO details using CMS Rest APIs
Notify employee on rejection by email

My Process

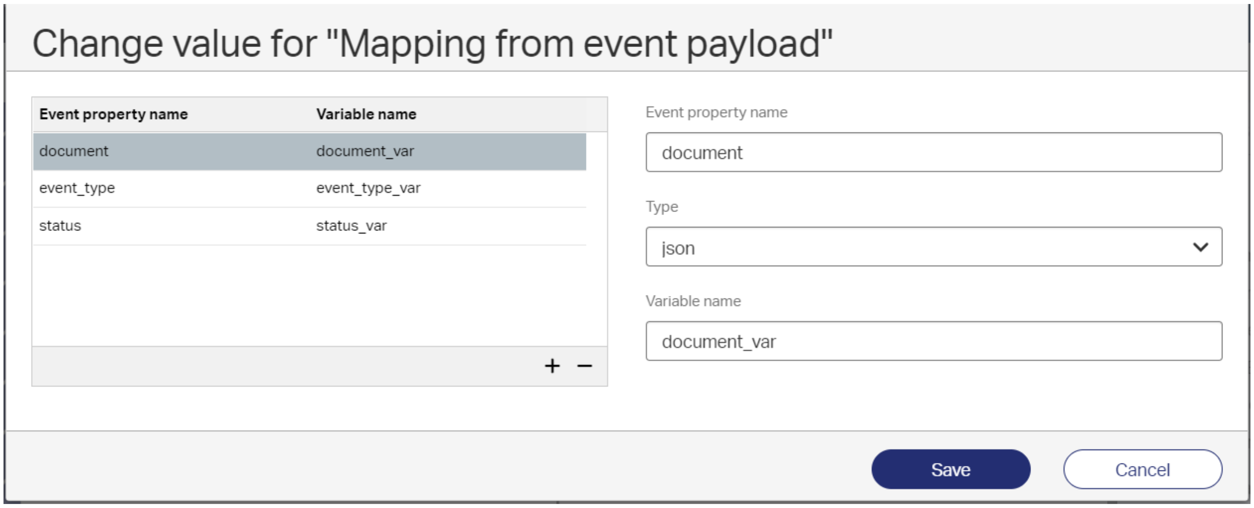
Id	sid-54163413-2796-40 ...
Name	No value
Documentation	No value
Message reference	No value
Cancel activity	☑
Event key	boundaryEvent
Event name	boundaryEvent
Mapping from event payload	2 event out parameter items
Correlation parameters	3 correlation parameters
Channel key	boundaryChannel
Channel name	boundaryChannel
Channel type	http
Event key fixed value	boundaryEvent
Event key json field	No value
Event key json pointer	No value

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Message reference	Name of the message.
Cancel activity	Cancels the activity, if selected.
Event Key	Key of the deployed event definition.
Event name	Name of the event definition.
Mapping from event payload	The required event payload values can be mapped to process variables.
Correlation parameters	When multiple process instances of current process definition are running, then the incoming event message can be matched against one of the running process instance.

Mapping from event payload :

Parameters can be configured to create required process variables from the event payload and can be used anywhere in the execution.



Change value for "Mapping from event payload"

Event property name	Variable name
document	document_var
event_type	event_type_var
status	status_var

Event property name: document

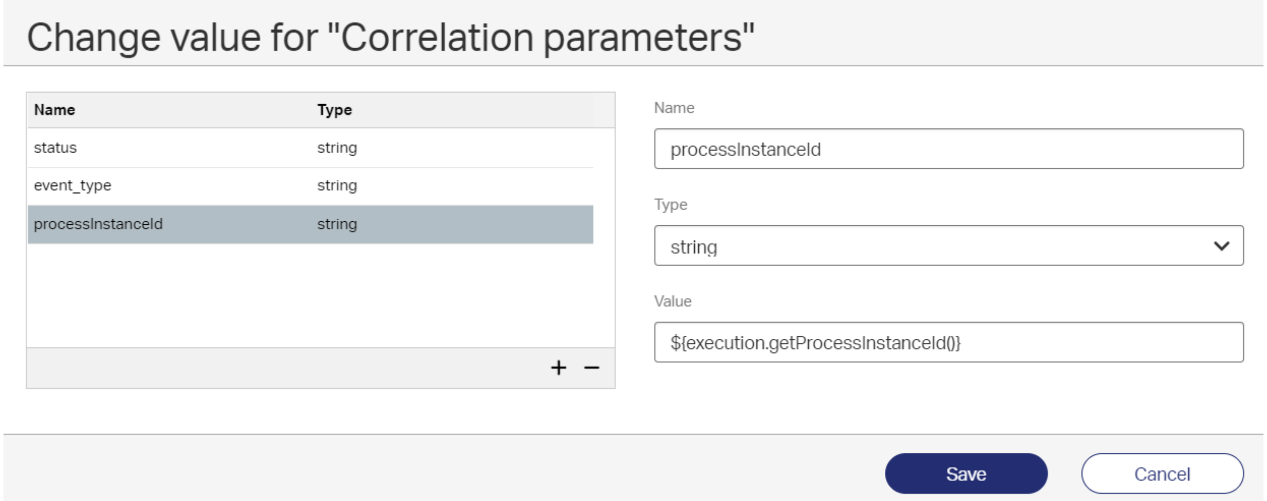
Type: json

Variable name: document_var

In above example three event parameters i.e document, event_type and status from event payload will be mapped to process variables document_var, event_type_var and status_var.

Correlation parameters :

Parameters can be configured to match the values against event payload, we could match multiple correlation parameters, if all of the correlation parameters are matched with the values of the received event payload then only the current activity will be cancelled and moves to the next activities.



Change value for "Correlation parameters"

Name	Type
status	string
event_type	string
processInstanceId	string

Name: processInstanceId

Type: string

Value: ${execution.getProcessInstanceId()}

In above example three correlation parameters i.e status, event_type and processInstanceId needs to matched exactly against the status, event_type and processInstanceId of the received event payload.

Here, we are using process instance id as a correlation parameter because when multiple process instances of current process definition are running, then the incoming event payload message can be matched against the current process instance.

End Events

An end event signifies the end of a path in a process or sub-process. An end event is always throwing a result. When process execution arrives at an end event, a result is thrown. The type of result is depicted by the inner black icon of the event. In the XML representation, the type is provided by the declaration of a sub-element.

None end event

A ‘none’ end event means that the result thrown when the event is reached is unspecified. As such, the business process engine will not perform anything besides ending the current path of execution.

Graphical notation

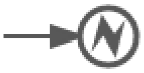
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.

End error event

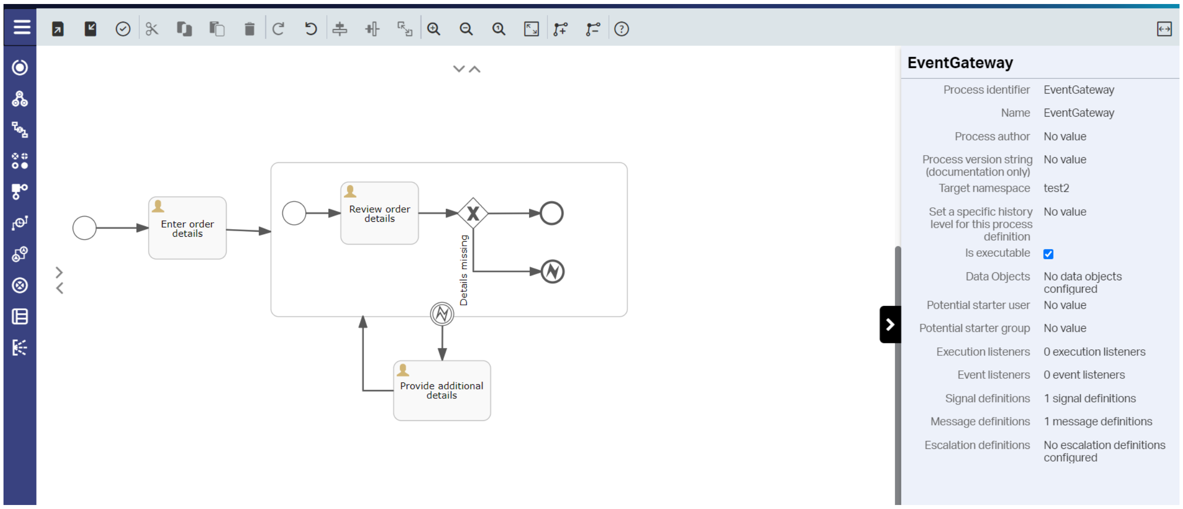
When process execution arrives at an error end event, the current path of execution ends and an error is thrown. This error can be caught by a matching intermediate boundary error event. If no matching boundary error event is found, an exception is thrown.

Graphical notation



Example

An end error event must be associated with an intermediate error catching event or boundary error event.



EventGateway

Process identifier	EventGateway
Name	EventGateway
Process author	No value
Process version string (documentation only)	No value
Target namespace	test2
Set a specific history level for this process definition	No value
Is executable	☑
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Execution listeners	0 execution listeners
Event listeners	0 event listeners
Signal definitions	1 signal definitions
Message definitions	1 message definitions
Escalation definitions	No escalation definitions configured

Following are the configurations.

- Check Is: A transaction sub-process check box.
- Check Is for compensation: A check box for an HTTP task, which is added at the boundary compensation event.

The model execution is as follows:

- A process instance is created.
- The Complete charge credit card task completes.
- Provide a condition at the gateway such that the cancel end event must trigger: (`${paid == false}`)
- Compensation occurs after the cancel end event.
- The boundary cancel event executes and the Send error details task releases.

Attributes

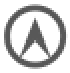
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.

Execution listeners	Active execution listeners of the activity. This lets you react to the following events: • Start: Occurs when the activity starts. • End: Occurs when the activity completes.
Error reference	Name of the error.

End escalation event

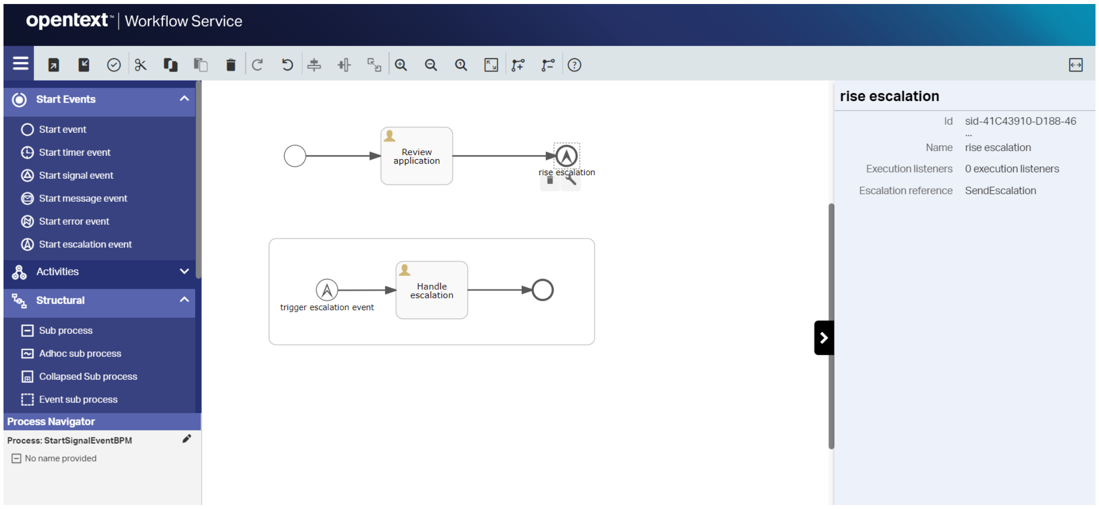
When process execution arrives at an escalation end event, the current path of execution ends and a named escalation is thrown. This escalation can be caught by an escalation boundary event or an event sub-process with an escalation start event.

Graphical notation



Example

An escalation end event must be configured with an escalation start event/boundary escalation event or escalation start event.



rise escalation

Id sid-41C43910-D188-46...
Name rise escalation
Execution listeners 0 execution listeners
Escalation reference SendEscalation

When Review application task completes, an escalation end event throws an escalation. The escalation start event catches the escalation and Handle escalation task releases.

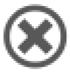
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity.
Escalation reference	Name of the escalation.

End cancel event

The cancel end event can only be used in combination with a transaction sub-process. When the cancel end event is reached, a cancel event is thrown, which must be caught by a cancel boundary event. The cancel boundary event cancels the transaction and triggers compensation.

Graphical notation



Example

Configuring and using the cancel event is explained in the Cancel boundary event.

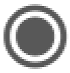
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity. This lets you react to the following events: • Start: Occurs when the activity starts. • End: Occurs when the activity completes.

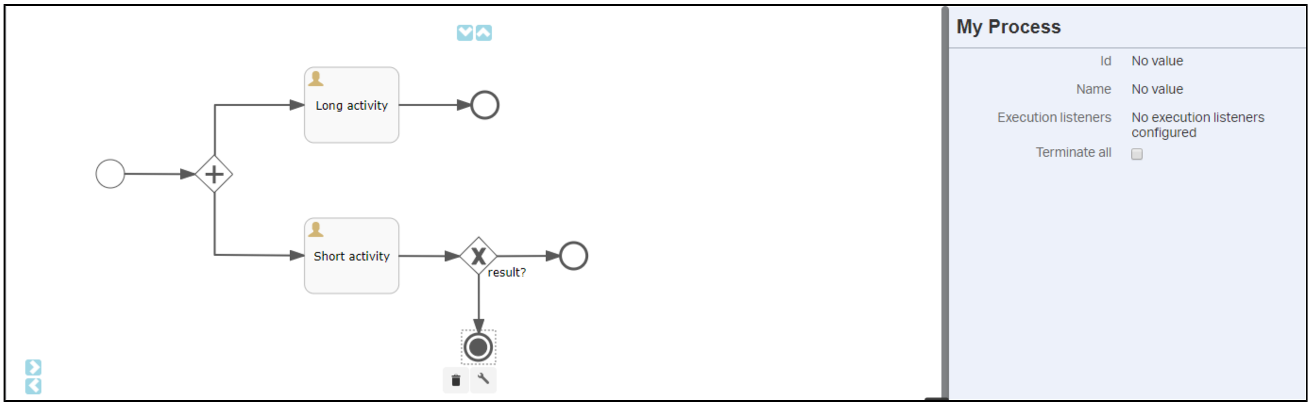
End terminate event

End terminate events are mostly used with parallel gateways. While a normal (untyped) end event indicates that a single process sequence ends, the terminate end event ends the whole process and thereby, ends every activity that may be running at that time.

Graphical notation



Example



My Process

Id No value
Name No value
Execution listeners No execution listeners configured
Terminate all □

When the upper end event is reached, only the first topmost sequence flow ends without considering the state of the other sequence flow. When the bottom activity ends, for example, with an outcome, which results in the fact that the upper activity is not needed anymore, the terminate end event ends both the parallel process flows, regardless of whether the top activity is still running.

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Execution listeners	Active execution listeners of the activity. This lets you react to the following events: • Start: Occurs when the activity starts. • End: Occurs when the activity completes.
Terminate all	Terminates the process instance, if enabled.

Swimlanes

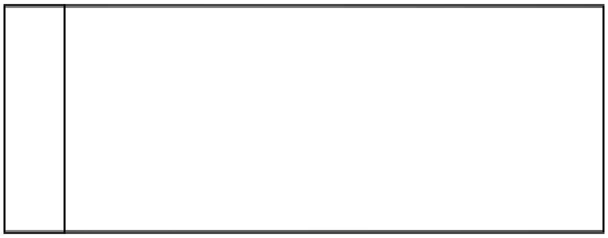
Swimlanes are rectangular boxes that represent participants of a business process. A swimlane may contain flow objects that are performed by that lane (participant). Swimlanes may be arranged horizontally or vertically. They are semantically the same but different in representation. For horizontal swimlanes, the process flows from left to right, while processes in vertical swimlanes flow from top to bottom.

There are two kinds of swimlanes: pools and lanes.

Pool

Pools represent participants in a business process. It can be a specific entity (for example, department) or a role (for example, assistant manager, doctor, student, or vendor). In a pool, there are flow elements, which represent the works that the pool must perform in the process being modeled.

Graphical notation



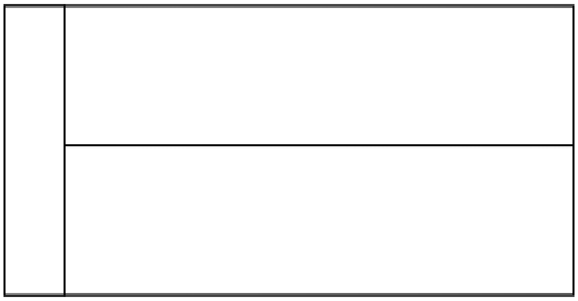
Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Process identifier	Unique identifier of the process definition.
is executable	Whether or not the process is executable.

Lane

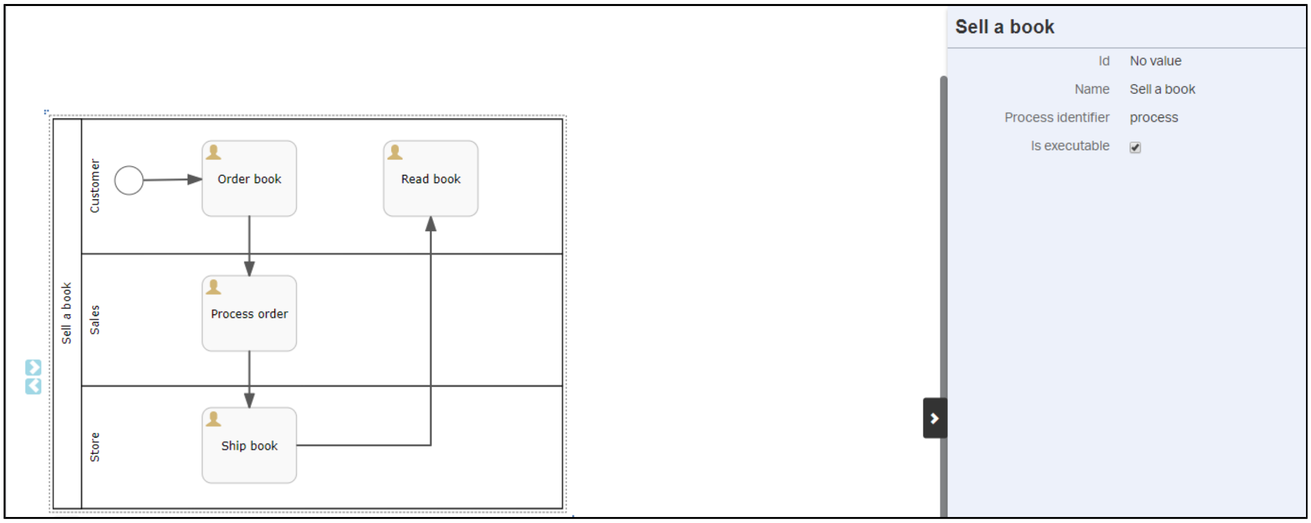
Lanes are sub-partitions of pools. For example, when you have a pool Department, you may have Department Head and General Clerk as lanes. Same as pools, you can use lanes to represent specific entities or roles who are involved in the process.

Graphical notation



Example (pool and lane)

The model shows the process of purchasing a book from a customer online. In this model, there is a pool (Sell a book) consisting of three lanes (Store, Sales, and Customer).



Sell a book

Id No value

Name Sell a book

Process identifier process

Is executable ☑

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.

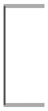
Artifacts

Artifacts allow you to visually represent objects outside the actual process. Artifacts can represent data or notes that describe the process, or they can be used to organize tasks or processes.

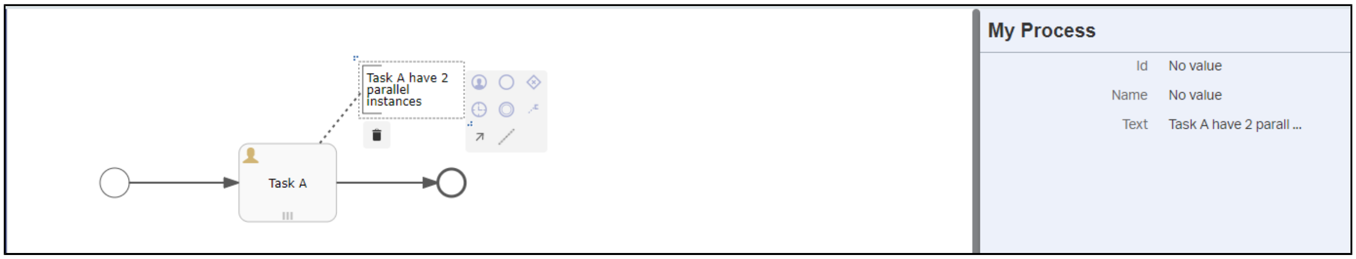
Text annotation

Annotations allow you to describe the business process and flow objects in more detail. Add annotations to make your BPMN process more readable and further increase understanding of your process.

Graphical notation



Example



My Process

Id No value

Name No value

Text Task A have 2 parall ...

The annotation for task A describes that task A has two instances and they are executed in parallel.

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.

Text	Text for the text annotation.

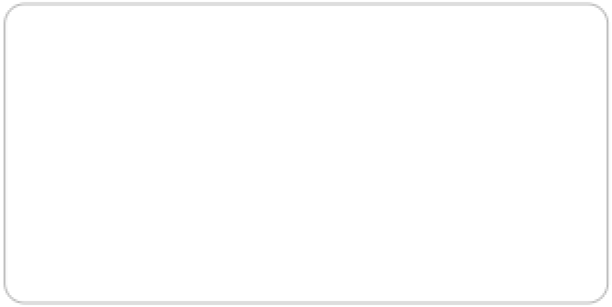
Structural

Sub-process

A sub-process is an activity that contains other activities, gateways, and events, which in itself form a process that is part of the bigger process. A sub-process is completely defined inside a parent process. A sub-process can only have one none start event. No other start event types are allowed. A sub-process must have at least one end event.

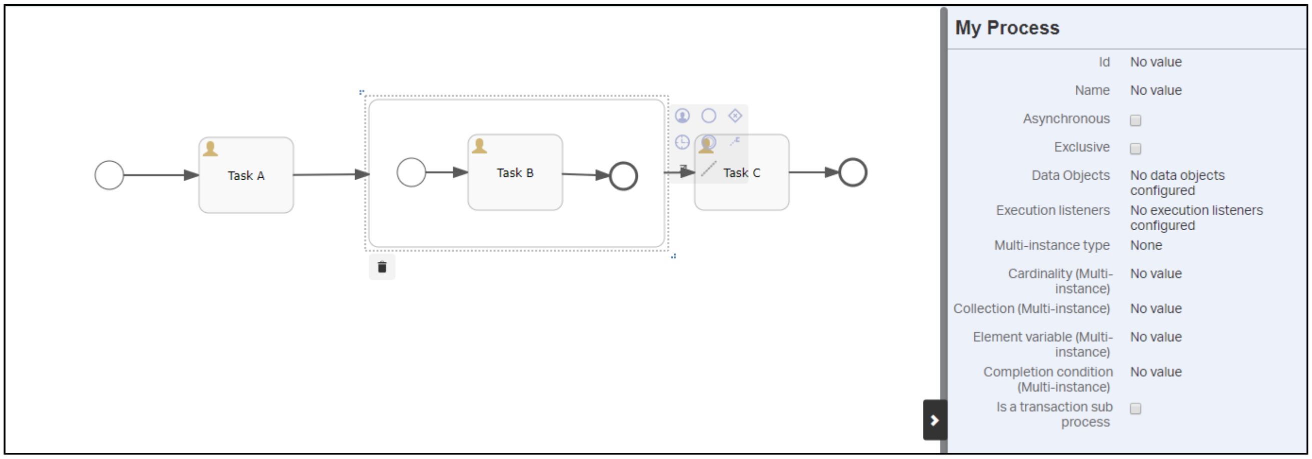
Note: The BPMN 2.0 specification allows the omission of the start and end events in a sub-process.

Graphical notation



Example

The following model shows a process with a subtask (Task B). After Task A completes, Task B initiates, and after completion of Task B, Task C starts. One of the main reasons to use a sub-process is to define a scope for a specific event.



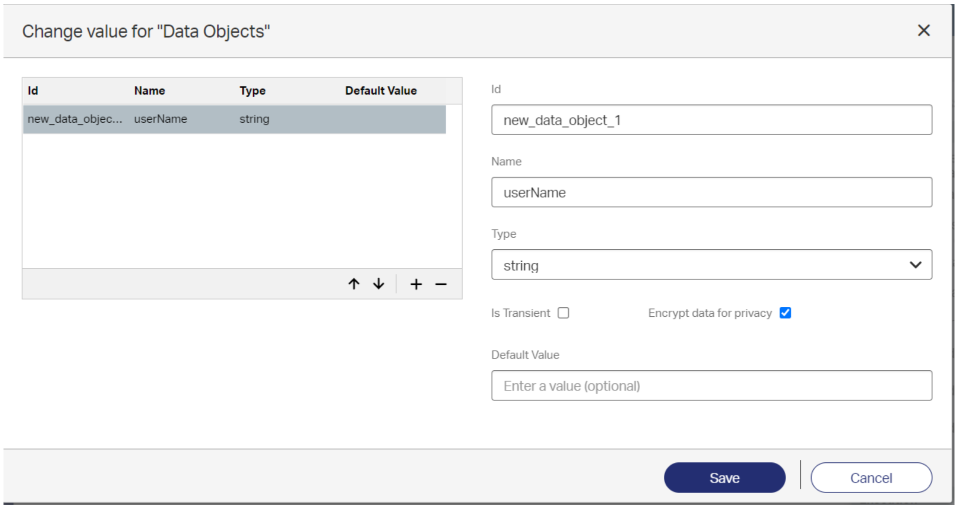
My Process

Id	No value
Name	No value
Asynchronous	☐
Exclusive	☐
Data Objects	No data objects configured
Execution listeners	No execution listeners configured
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is a transaction sub process	☐

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When enabled, the activity is started as an asynchronous job. The process state persists before this element executes. Then, the process execut if an error occurs before the following wait state, there is no direct user feedback.
Exclusive	Defines the activity as exclusive.
Execution listeners	Active execution listeners of the activity. This lets you react to the following events: • Start: Occurs when the activity starts. • End: Occurs when the activity completes.

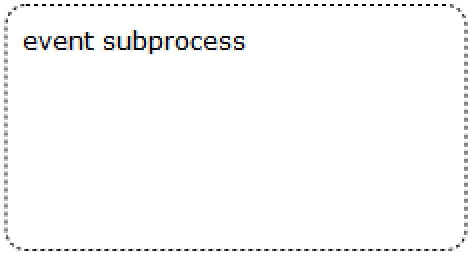
Multi-Instance type	Whether this task is performed multiple times and how it is performed. The possible values are: • None: The task is performed once only. • Parallel : The task is performed multiple times with each instance potentially occurring at the same time as the others. • Sequential: The task is performed multiple times, one instance following on from the previous one.
Cardinality (Multi-instance)	Number of times to perform the task.
Collection (Multi-instance)	Name of a process variable, which is a collection. For each item in the collection, an instance of this task is created.
Element variable (Multi-instance)	Process variable name, which contains the current value of the collection in each task instance.
Completion condition (Multi-instance)	A multi-instance activity normally ends when all instances end. Specify an expression to evaluate each time an instance ends. If the expression e
Is a transaction subprocess	Whether this sub-process is of type transaction.

Data Objects	Definition of data object properties. Encrypt data for privacy :  Change value for "Data Objects" ✕ Id / Name / Type / Default Value new_data_objec... / userName / string / Id: new_data_object_1 Name: userName Type: string Is Transient □ Encrypt data for privacy ☑ Default Value: Enter a value (optional) Save Cancel Use this option to encrypt the data used in data objects. By default, data objects are not selected for encryption.

Event sub-process

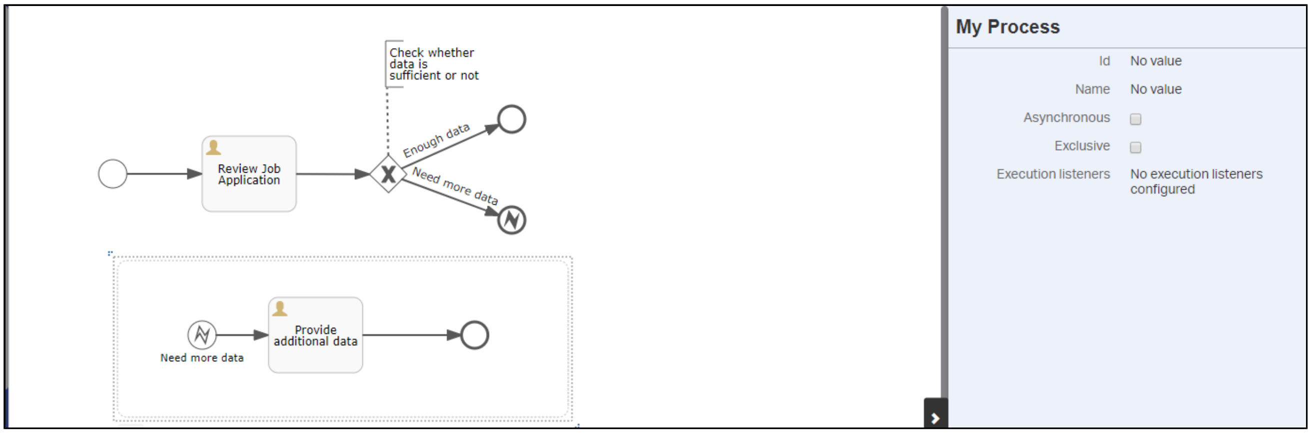
An event sub-process is triggered by an event. It can be added at the process level or any sub-process level. The event used to trigger an event sub-process is configured using a start event, implying that none start events are not supported for event sub-processes. An event sub-process might be triggered using events, such as message events, error events, signal events, timer events, or compensation events. An event sub-process must not have any incoming or outgoing sequence flows because an event sub-process is triggered by an event.

Graphical notation



Example

The following model shows a job application review process. In the review task, the process checks for the required information, and if the information is not sufficient, an error event is thrown and the error event in the event sub-process is triggered.



My Process

Id: No value
Name: No value
Asynchronous □
Exclusive □
Execution listeners: No execution listeners configured

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When enabled, the activity is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This can be used when the execution of an activity takes a long time to return the user interface. However, if an error occurs before the following wait state, there is no direct user feedback.
Execution listeners	Active execution listeners of the activity. This lets you react to the following events: • Start: Occurs when the activity starts. • End: Occurs when the activity completes.
Exclusive	Defines the activity as exclusive.

Call activity

BPMN 2.0 makes a distinction between a regular sub-process, often also called embedded sub-process, and the call activity, which looks very similar. Both call a sub-process when the process execution arrives at the activity.

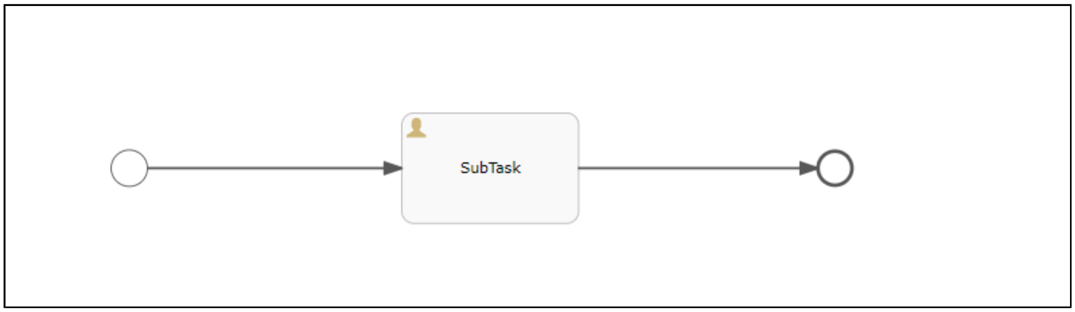
The difference is that the call activity references a process that is external to the process definition, whereas the sub-process is embedded within the original process definition. The main use case for the call activity is to have a reusable process definition that can be called from multiple other process definitions.

When process execution arrives at the call activity, a new execution is created that is a sub-execution of the execution that arrived at the call activity. This sub-execution is then used to execute the sub-process, potentially creating parallel child executions as within a regular process. The super-execution waits until the sub-process ends and continues with the original process afterward.

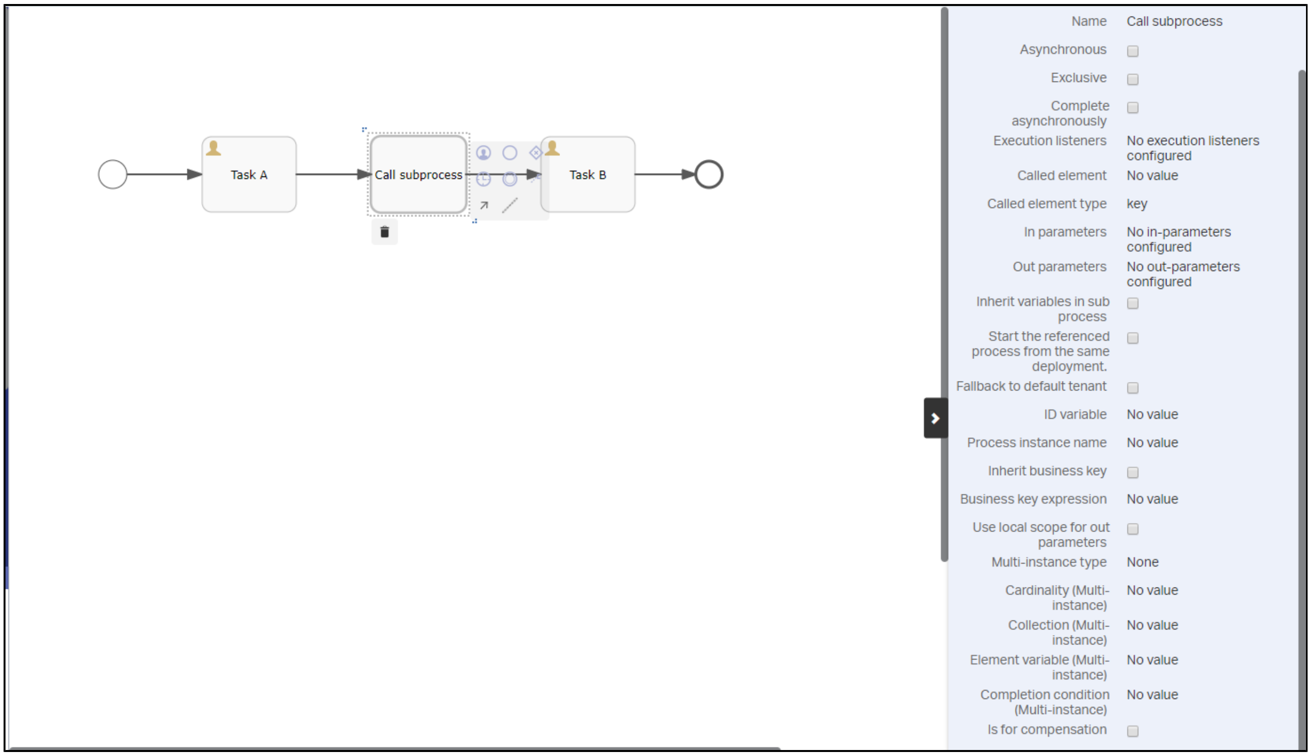
Graphical notation

Example

A sub-process model:



The main process model utilizing call activity:



Name	Call subprocess
Asynchronous	□
Exclusive	□
Complete asynchronously	□
Execution listeners	No execution listeners configured
Called element	No value
Called element type	key
In parameters	No in-parameters configured
Out parameters	No out-parameters configured
Inherit variables in sub process	□
Start the referenced process from the same deployment.	□
Fallback to default tenant	□
ID variable	No value
Process instance name	No value
Inherit business key	□
Business key expression	No value
Use local scope for out parameters	□
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is for compensation	□

In the call activity properties, select the key value for the Called element type and provide the sub-process key name in the Called element.

Attributes

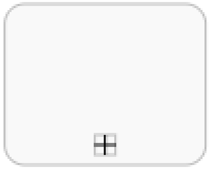
Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Start the referenced process from the same deployment	References the referenced process from the same application deployment, if set to `true`. Set to `false` to always use the newest process definition.
Called element type	Key or ID of the deployed process definition to start the referenced process.
Called element	Called element value.
In parameters	Optional input parameter map. Allows the mapping of parameters and variables, which are then available in the newly created process.
Out parameters	Optional output parameter map. Allows the mapping of parameters and variables to the original case work item when the human task's work item completes.
Inherit variables in subprocess	Inheritance of parent process variables in the sub-process.
Process instance name	Expression that resolves into the name of the child process instance.
ID variable	Instance ID of the started instance is store, if set.
Inherit business key	Inheritance of the business key from the parent process.
Business key expression	Business key of the newly created process instance, which can be an expression.
Use local scope for out parameters	Uses local variable scope for out parameters.
Complete asynchronously	Executes the completion of the call activity in an exclusive asynchronous job. This is useful in combination with parallel multi-instance.

Fallback to default tenant	Indicates that the process instance is created with the default tenant if it is not available on the current tenant, if the application is running in a multi-tenant setup.
Execution listeners	Active execution listeners of the activity. This lets you react to the following events: • Start: Occurs when the activity starts. • End: Occurs when the activity completes.
Asynchronous	When enabled, the activity is started as an asynchronous job. The process state persists before this element executes, and the process execution resumes asynchronously. This can be used when the execution of an activity takes a long time to return the user interface. However, if an error occurs before the following wait state, there is no direct user feedback.
Is for compensation	Whether the activity can serve as a compensation for another activity.
Exclusive	Defines the activity as exclusive.
Multi-instance type	Whether this task is performed multiple times and how it is performed. The possible values are: • None: The task is performed once only. • Parallel : The task is performed multiple times with each instance potentially occurring at the same time as the others. • Sequential: The task is performed multiple times, one instance following on from the previous one.
Cardinality (Multi-instance)	Number of times to perform the task.
Collection (Multi-instance)	Name of a process variable, which is a collection. For each item in the collection, an instance of this task is created.
Element variable (Multi-instance)	Process variable name, which contains the current value of the collection in each task instance.
Completion condition (Multi-instance)	A multi-instance activity normally ends when all instances end. Specify an expression to evaluate each time an instance ends. If the expression evaluates to `true`, all remaining instances are destroyed and the multi-instance activity ends.

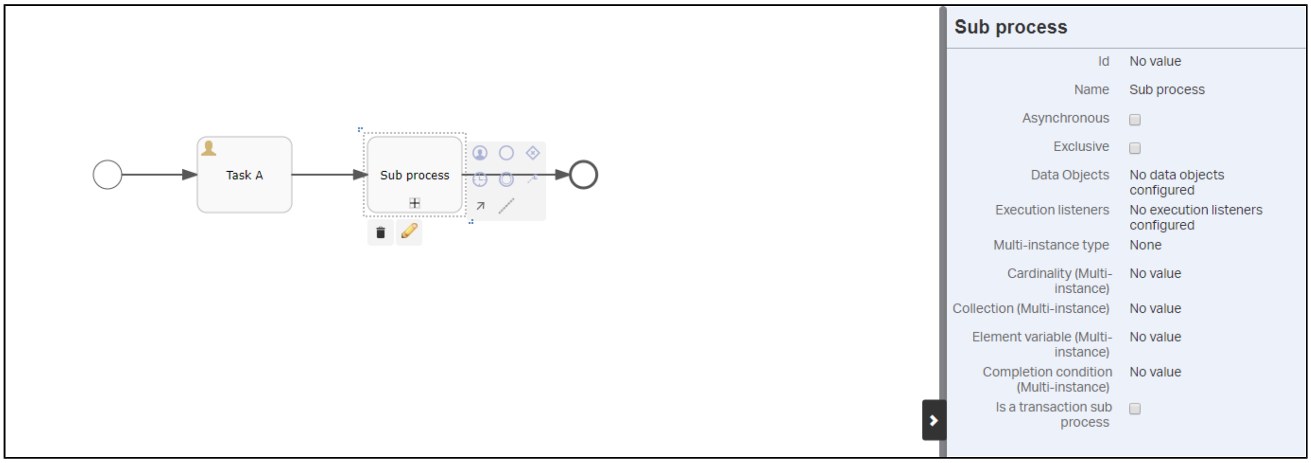
Collapsed sub-process

Many modeling tools allow sub-processes to be collapsed, hiding all the details of the sub-process, resulting in a high-level, end-to-end overview of the business process. A sub-process is visualized as a typical activity (a rounded rectangle). If the sub-process is collapsed, only the name and a plus-sign are displayed, providing a high-level overview of the process.

Graphical notation



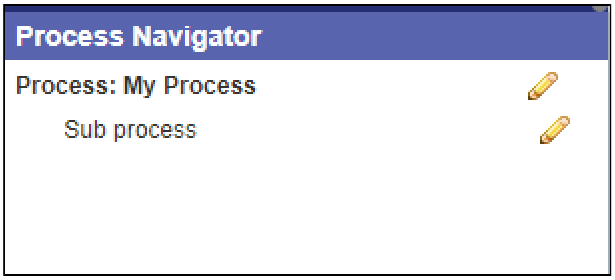
Example



Sub process

Id	No value
Name	Sub process
Asynchronous	□
Exclusive	□
Data Objects	No data objects configured
Execution listeners	No execution listeners configured
Multi-instance type	None
Cardinality (Multi-instance)	No value
Collection (Multi-instance)	No value
Element variable (Multi-instance)	No value
Completion condition (Multi-instance)	No value
Is a transaction sub process	□

You can switch between the main process and sub-process using the process navigator in the modeler.



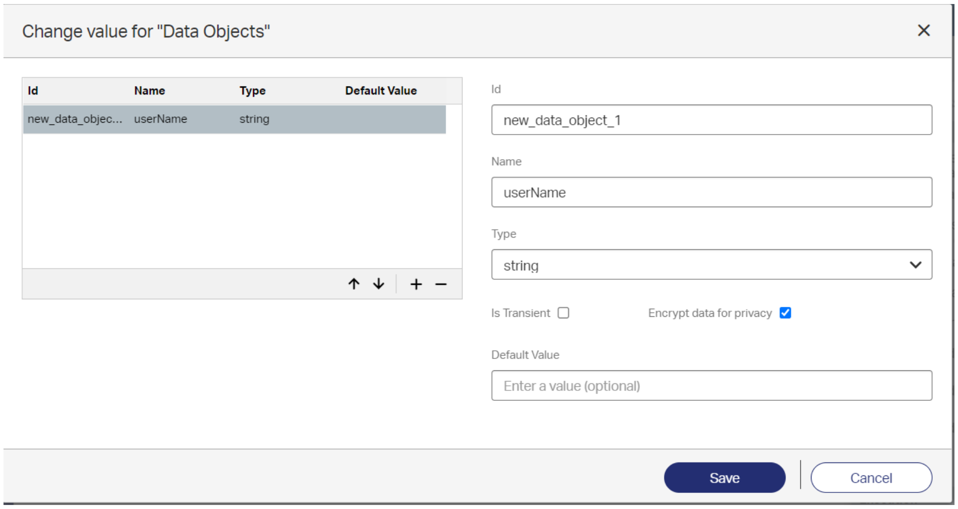
Process Navigator

Process: My Process

Sub process

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.
Asynchronous	When enabled, the activity is started as an asynchronous job. The process state persists before this element executes, and the process executio an error occurs before the following wait state, there is no direct user feedback.
Execution listeners	Active execution listeners of the activity. This lets you react to the following events: • Start: Occurs when the activity starts. • End: Occurs when the activity completes.
Multi-Instance type	Whether this task is performed multiple times and how it is performed. The possible values are: • None: The task is performed once only. • Parallel : The task is performed multiple times with each instance potentially occurring at the same time as the others. • Sequential: The task is performed multiple times, one instance following on from the previous one.
Cardinality (Multi-instance)	Number of times to perform the task.
Collection (Multi-instance)	Name of a process variable, which is a collection. For each item in the collection, an instance of this task is created.
Element variable (Multi-instance)	Process variable name, which contains the current value of the collection in each task instance.
Completion condition (Multi-instance)	A multi-instance activity normally ends when all instances end. Specify an expression to evaluate each time an instance ends. If the expression e
Is a transaction subprocess	Whether this sub process is a type of transaction.

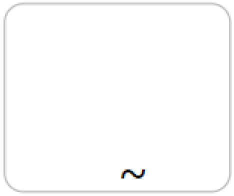
Data Objects	Defination of data objects properties. Encrypt data for privacy :  Change value for "Data Objects" Id | Name | Type | Default Value new_data_objec... | userName | string | Id: new_data_object_1 Name: userName Type: string Is Transient ☐ Encrypt data for privacy ☑ Default Value: Enter a value (optional) Use this option to encrypt the data used in data objects. By default, data objects are not selected for encryption.
Exclusive	Defines the activity as exclusive.

Adhoc sub-process

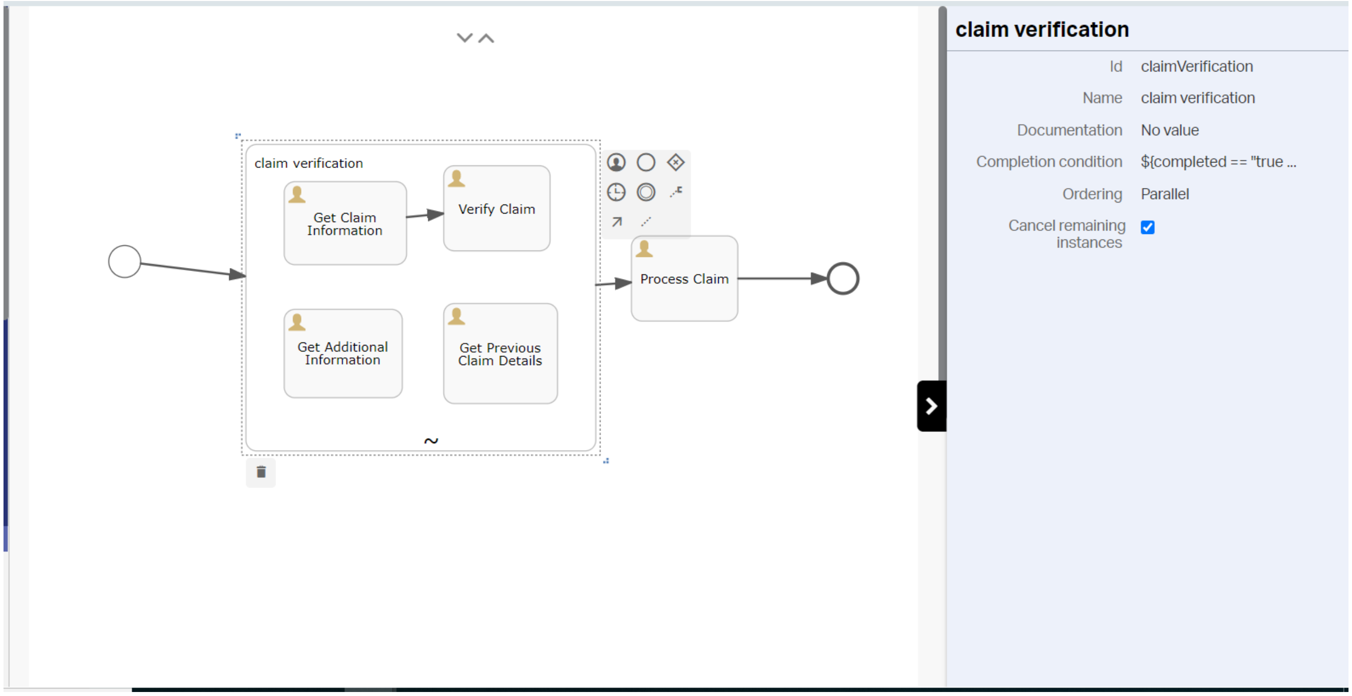
Use the ad-hoc subprocess to mark a segment in which the contained activities can be:

- Executed in any order,
- Executed several times, or
- Skipped.

Graphical notation



Example



claim verification

Id claimVerification

Name claim verification

Documentation No value

Completion condition ${completed == "true ...

Ordering Parallel

Cancel remaining instances ☑

- When a process instance is started from this process definition, an ad-hoc subprocess execution is created in the workflow engine and it doesn't execute the child activities automatically.
- To execute the activities in the ad-hoc process, Execute workflow rest API '/process-instances/adhoc-subprocesses/{adhocSubprocessId}/activities'
  Example request body:

  ```
  {
         "activityIds": [
          "GetClaimInformation", "GetPreviousClaimDetails"
          ]
  }
  ```

  To execute this API, It requires ad-hoc subprocess execution Id, activity Id's in the ad-hoc subprocess.

  - To get the Id of ad-hoc subprocesses for a process instance, Execute workflow rest API '/process-instances/{processInstanceId}/adhoc-subprocesses'.

  - To get the List of enabled activities for ad-hoc subprocess, Execute workflow rest API '/process-instances/adhoc-subprocesses/{adhocSubprocessId}/activities'.

- Without defining a 'completion condition' attribute, the Workflow Engine will not end the ad-hoc subprocess execution automatically. To complete an ad-hoc subprocess, Execute workflow rest API '/process-instances/adhoc-subprocesses/{adhocSubprocessId}' when there are no active child executions (for example user tasks) anymore.
  Example request body:

  ```
  {
          "action": "complete"
  }
  ```

Attributes

Attribute	Description
ID	Unique identifier of the element within the process model.
Name	Name of the element. This is the name displayed in the diagram.

Completion condition	Specify an expression to evaluate each time while completing a child execution. If the expression evaluates to true and if the Cancel remaining instances attribute is set to true then the ad-hoc subprocess will be completed automatically. If the expression evaluates to true and if the cancel remaining instances attribute set to false then the ad-hoc subprocess will be completed only when there are no active child executions. Without defining a completion condition expression the workflow engine will not end the ad-hoc subprocess execution automatically. Need ad-hoc subprocess complete API to complete the sub-process when there are no active child executions (for example user tasks) anymore.
Ordering	▪ Parallel: We can execute multiple enabled activities at the same time. By default, this attribute set to Parallel. ▪ Sequential: This means that only one of the activity can be executed at the same time. The workflow engine will not allow a second activity to be executed when the first activity hasn't been completed yet.
Cancel remaining instances	It's possible to define whether the ad-hoc sub process should cancel any remaining executions when the completion condition evaluates to true. By default it is true, the Workflow Engine will cancel all other running executions, but when setting this attribute to false, the ad-hoc sub process will not complete before all executions have been ended.