

ADP Workforce Manager

Business Processes Developer's Guide



The information contained in this document is intended for use with the licensed software products to which the information relates (the "Products"). The information and the specifications for the Products set forth herein are subject to change without notice, and should not be construed as a commitment by the licensor to provide the functionality described herein. The licensor may make improvements and/or changes to the Products and/or the information set forth in this publication at any time without notice. The licensor assumes no responsibility for any errors that may appear in this resource. © 2023 UKG Inc. All rights reserved.

This document contains certain trademarks that are the property of UKG Inc., which may be found on the "trademarks" page at <http://www.ukg.com>. All other trademarks or registered trademarks used herein are the property of their respective owners and are used for identification purposes only.

When using and applying the information generated by the Products, customers should ensure that they comply with the applicable requirements of all applicable law, including federal and state law, such as the Fair Labor Standards Act. Nothing in this document shall be construed as an assurance or guaranty that the Products comply with any such laws.

Published by ADP, Inc.

One ADP Boulevard

Roseland, New Jersey 07068

For more information, see the following ADP Web page: <http://www.adp.com>

Document Revision History

Document Revision	Release Date
A	September 2023

Contents

Chapter 1: Overview	9
Terminology	9
Business Processes Used in ADP Workforce Manager	10
Workflow Designer workspace components	11
Process Display view	12
Decision Tables view	12
Visual Editor	13
Create a Business Process model	14
Step 1: Open Workflow Designer	14
Step 2: Configure process parameters or variables	16
Step 3: Configure task parameters	17
Step 4: Link the events	17
Step 5: Finish and verify the process	18
Step 6: Deploy the Model	19
Business Process Best Practices	20
Chapter 2: Start Events	25
None start event	25
Start timer event	26
Start signal event	27
Start error event	27
Chapter 3: Activities	29
User tasks	29

Script tasks	32
Example initialization of an assigneeList process variable	36
Mail tasks	38
Decision tasks	40
API service tasks	44
Generic Rest Connector	48
Key elements	52
Internal REST Connector	55
Initialize Variables task	59
Attestation Text Substitution	67
Configure automatic comments for manual time entry Attestation workflow models	68
Chapter 4: Structural	73
Subprocess	73
Collapsed Subprocess	75
Embedded subprocess	76
Event subprocess	78
Call activity subprocess	78
Configure a Call Activity subprocess	80
Execute Business Process By ID	81
Submit User Task By ID	82
Chapter 5: Gateways	83
Exclusive gateway	83
Parallel gateway	88
Inclusive gateway	89

Event gateway	90
Example:	91
Chapter 6: Events	93
Boundary Events	93
Intermediate Catching Events	94
Intermediate Throwing Events	95
End Events	95
Example: Add an escalation timer and signal event to a workflow	95
Listeners	101
Chapter 7: Artifacts	105
Chapter 8: APIs	107
Chapter 9: Create Forms	109
Form editor stencil pallet	110
Example: Create a form	118
Chapter 10: Custom Beans in Business Process	121
Preconfigured custom beans	121
Use of custom beans in an expression	121
Custom beans for custom extensions	122
Chapter 11: Business Processes for Devices	123
Variables	123
Forms	123
Form name	124
Container	124

Field types and transformations	124
Form with select options	125
Chapter 12: Integrations in a Business Process	127
Run an integration or integration set	128
Schedule an integration or integration set	128
Chapter 13: Exception Handling	131
Methods to handle exceptions	131
Boundary event error handler	131
Event subprocess	132
Scripts	133
Transaction boundaries	134
Compensation	134
Configure exception handling	135
Error handling from a script task	135
Error handling from a gateway	137
Error handling from a Generic REST Connector or service task	141
Chapter 14: Business Process Errors	143
Chapter 15: Troubleshoot Business Processes	147
Use the Business Process Admin	147
Log Business Processes in ADP Workforce Manager	147
Logged information	148
Start logging	148
Stop logging	149

Display a log	149
Export a log	150

Chapter 1: Overview

ADP Workforce Manager includes a Business Process development tool called Workflow Designer. This graphical tool enables developers to define Business Processes that include human tasks and service calls and then execute the tasks in a defined order, while exposing the API to start, manage, and query data about the process.

Workflow Designer is based on the Alfresco Activiti Business Process management (BPM) engine which adheres to the Business Process Model and Notation (BPMN) standard for Business Process diagrams.

This document describes how developers can edit and create Business Process workflows within ADP Workforce Manager, which can then be exposed to users in the ADP Workforce Manager user interface.

Terminology

The following workflow entity terms are used throughout this document:

- **Process models** – A BPMN 2.0 standard design templates or graphical representation of Business Processes. A number of default process models are available from the ADP Workforce Manager Process Models setup page. Developers can edit these process models or create new ones in Workflow Designer.
- **Business Processes** – Organized workflows of business activities or tasks that achieve a business goal. A Business Process is created when a process model is deployed from the Process Model setup page.
- **Process definitions** – BPMN 2.0 standard XML files that are created and stored in the workflow engine when a process model is deployed. Process definitions can be executed to run business workflows.
- **Process instances** – A running instance of a process definition. After a process instance has been started, its associated process definition cannot be changed. If you change the underlying process definition, a new version is created but the already-running instance continues to refer to the older version of the process definition.

Only new process instances use the changes from an updated process definition version.

When the server is stopped and restarted, process instances including user tasks as well as process and execution variables are unchanged.

- **Tasks** – Not all Activiti tasks are included in ADP Workforce Manager. Workflow Designer is limited to the following tasks:
 - **userTask** – Work to be done by a human actor. When process execution arrives at a user task, a new task is created in the task list of the user or group assigned to that task.
 - **scriptTask** – An automatic activity. When a process execution arrives at the script task, the corresponding script is executed.
 - **mailTask** – Similar to a script task, but is specifically set up to send an email.
 - **API Service Tasks** – Customized versions of service tasks, which are categorized separately under the **APIs** label on the stencil set.
- **Jobs** – Jobs are used in workflows for timers, asynchronous continuations, delayed suspension/activation, and so forth. When you execute a process definition with a timer event, jobs are created together with the process instance.
- **Initialization variables** – Data used to complete the execution of a workflow. Variables may be associated with a specific step or task in the workflow or they may be used globally across all steps or tasks in the workflow. Variables, which are stored in the workflow database, can be used in many ways including the following:
 - As expressions, for example to select the correct outgoing sequence flow in an exclusive gateway
 - As Java service tasks when calling external services, for example to provide the input or store the result of the service call

Business Processes Used in ADP Workforce Manager

A number of components within ADP Workforce Manager use Business Processes to complete specific tasks, for example:

- **Control Center** – Uses Task notifications to interact with workflow forms. Forms can be submitted and assigned to different users based on business needs. Examples are manager and administrator delegation.
- **Time-Off Requests** – Uses Business Processes to add various actions to time-off requests. Examples are resume API task, automatic action, and reminder.
- **UDM** – The UDM QuickGlance broker integrates a device with a QuickGlance process defined in

Workflow Designer, allowing a process instance to be presented and controlled on a device.

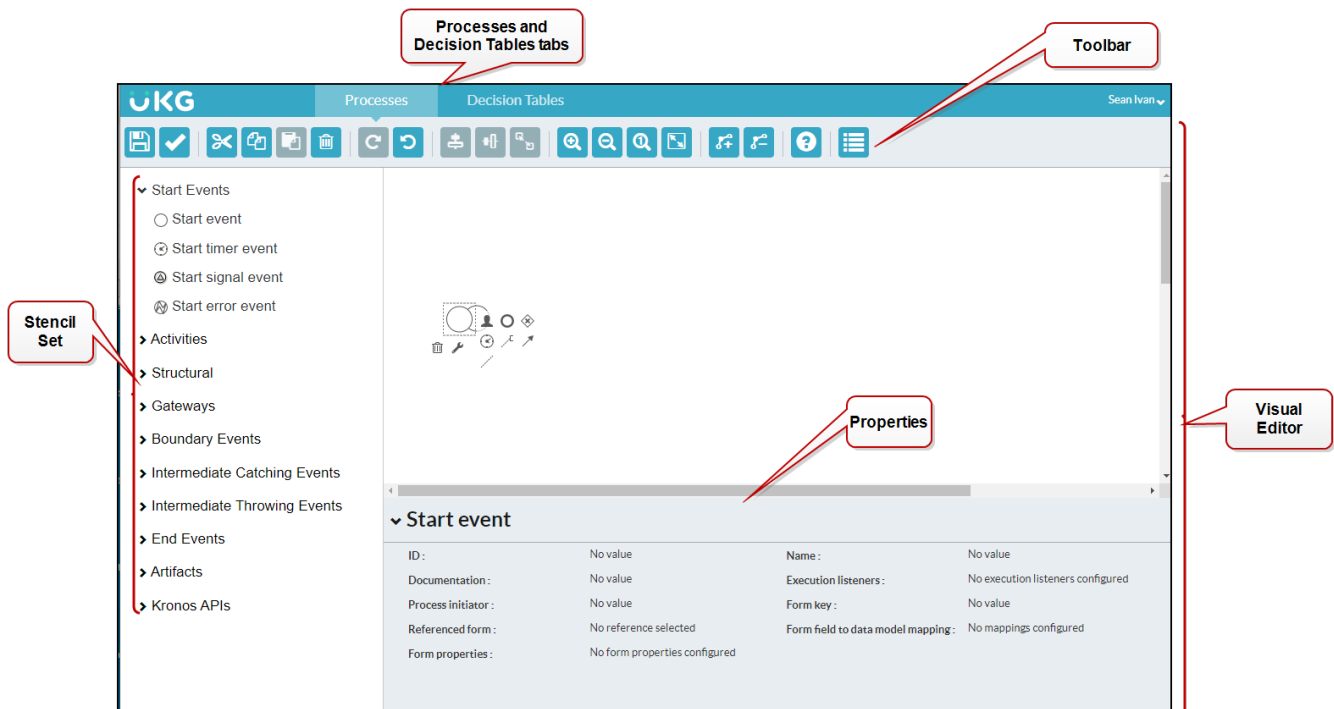
UDM acts only as a pass-through between a device and a process; all QuickGlance form definitions, business logic, API integration, and flow occur in the process. This allows the QuickGlance to be entirely implemented in Workflow Designer.

- **Attestation** – Uses Business Processes to configure conditional workflows that meet customer compliance policies. For example, Daily Attestation and Meal Lockout workflows.
- **Business Process tile** – Initiates a Business Process from a Home Page tile.
- **GoTo link and the Business Process Library** – Initiates a Business Process.

Workflow Designer workspace components

The main components of the Workflow Designer workspace are:

- [Process Display view on page 12](#)
- [Decision Tables view on page 12](#)
- [Visual Editor on page 13](#)



Process Display view

When you click the **Processes** tab at the top of Workflow Designer, the Process Display opens. The Process Display includes the following links as well as search and filter tools.

- **My Processes** – Lists the process models created by logged-in user.
- **Shared with Me** – Lists the process models created by other users of the same tenant.
- **Shared with others** – Lists the process models created by the logged-in user that are ready to be used by other users of same tenant.

You also access the Process Display when you select a model and click **Edit** on the Process Models Setup page.

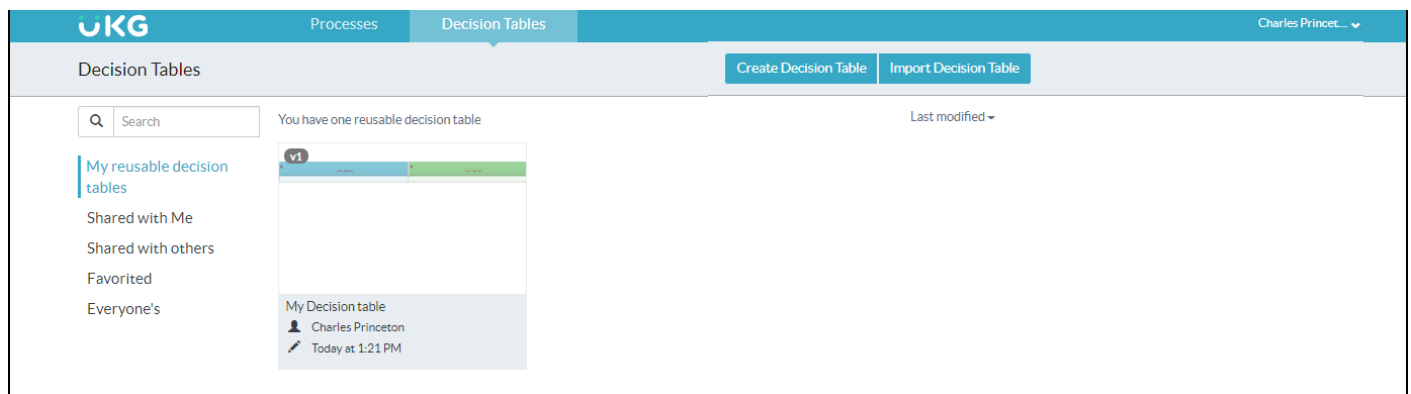
Decision Tables view

Decision tables enable the user to make a decision on the basis of a set of conditions during the execution of a Business Process. The user can define input and output variables and decide on their values on the basis of the rules defined in the table. The variables defined as the input are used to define the conditions and the variables defined as the output are the results of the conditions defined.

These variables can be further used in the model and their values are populated dynamically during the execution of the Business Process based on the rules defined in the decision table.

The decision table can be created/accessed through the Decision task in the stencil set as well as through the Decision Tables tab. You can also import, export, delete, and copy these tables from the **Decision Tables** tab.

For more information, see [Decision Tables](#).

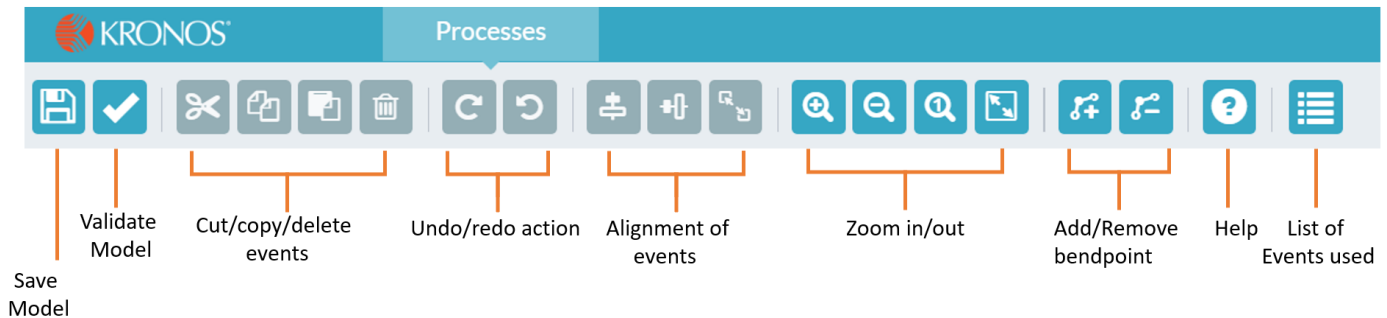


Visual Editor

The Visual Editor is where you create and edit Business Processes. It contains a toolbar, the stencil set, and a canvas.

Toolbar

Located along the top of the Visual Editor workspace, the toolbar is a graphical control element for quick access to functions used by Workflow Designer.



Stencil Set

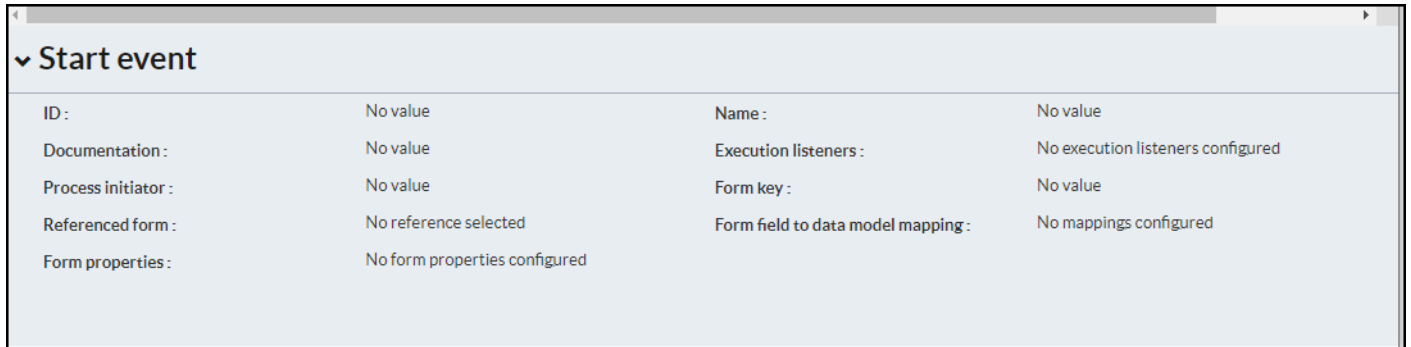
The stencil set, which is displayed along the left side of the Visual Editor, contains the events that you use to create a process. The categories of these events are:

- [Start Events on page 25](#)
- [Activities on page 29](#)
- [Structural on page 73](#)
- [Gateways on page 83](#)
- [Boundary Events on page 93](#)
- [Intermediate Catching Events on page 94](#)
- [Intermediate Throwing Events on page 95](#)
- [End Events on page 95](#)
- [Artifacts on page 105](#)
- [APIs on page 107](#)

Much of this document is organized by these stencil set groups.

Canvas

The canvas is the space where the Business Process is assembled. When you drag an item from the stencil set to the canvas, the properties associated with that item are displayed at the bottom of the workspace. For example, the following properties are available for the Start event.



The screenshot shows a properties panel for a 'Start event'. The panel has a title bar with a dropdown arrow and the text 'Start event'. Below the title bar, there are several rows of properties, each with a label on the left and a value on the right. The properties are: ID (No value), Name (No value), Documentation (No value), Execution listeners (No execution listeners configured), Process initiator (No value), Form key (No value), Referenced form (No reference selected), and Form field to data model mapping (No mappings configured). The last row, 'Form properties', has a value of 'No form properties configured'.

Start event			
ID :	No value	Name :	No value
Documentation :	No value	Execution listeners :	No execution listeners configured
Process initiator :	No value	Form key :	No value
Referenced form :	No reference selected	Form field to data model mapping :	No mappings configured
Form properties :	No form properties configured		

When you select a property, additional dialog boxes and pages open to help you configure the property.

Create a Business Process model

Business Process models can vary significantly depending on scope and complexity, but the general steps are:

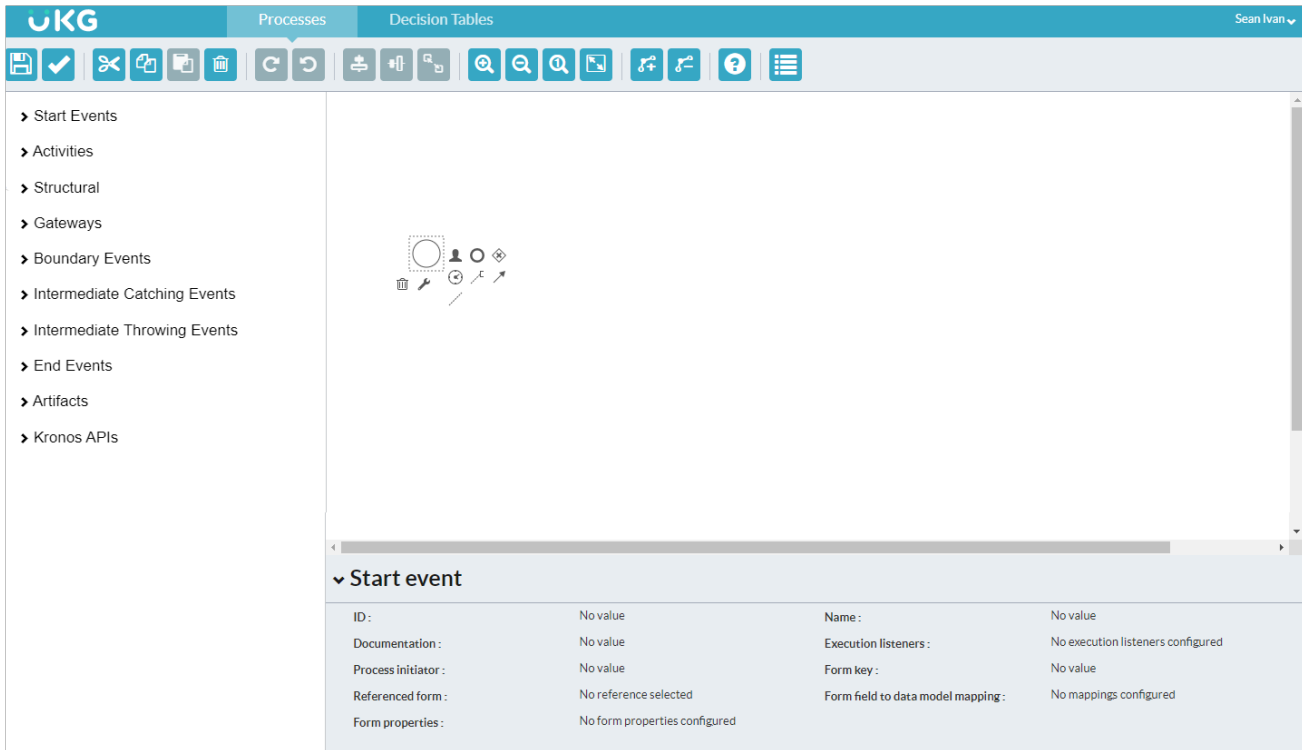
- [Step 1: Open Workflow Designer on page 14](#)
- [Step 2: Configure process parameters or variables on page 16](#)
- [Step 3: Configure task parameters on page 17](#)
- [Step 4: Link the events on page 17](#)
- [Step 5: Finish and verify the process on page 18](#)
- [Step 6: Deploy the Model on page 19](#)

Step 1: Open Workflow Designer

To open Workflow Designer from the ADP Workforce Manager user interface:

1. Click **Main Menu**  > **Administration** > **Application Setup** > **Business Process Setup** > **Process Models**.

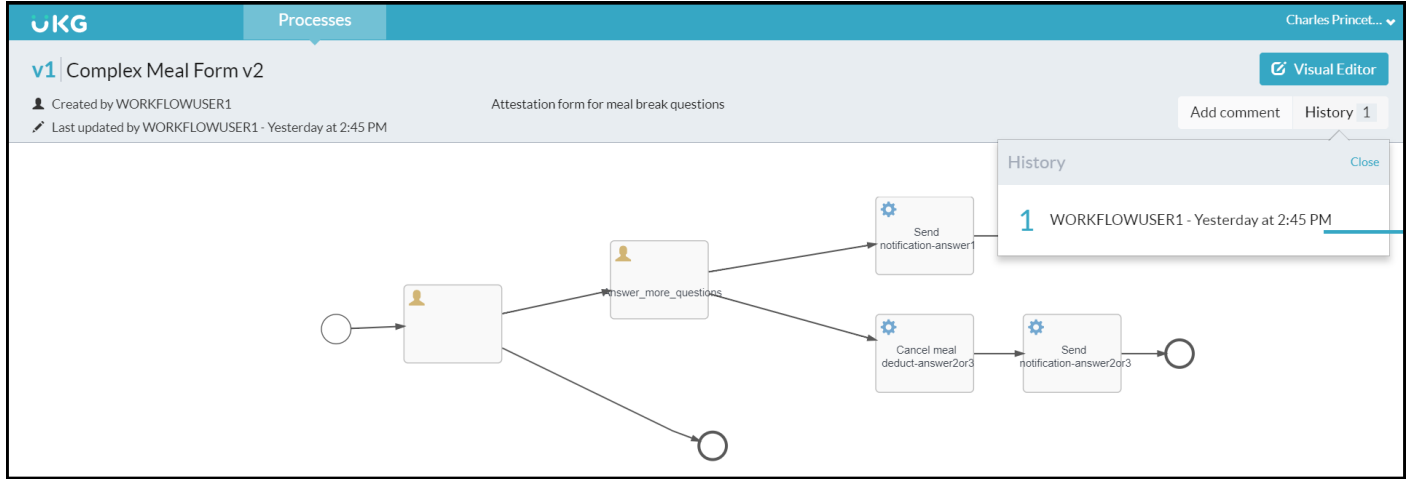
2. Click **Create**, then enter a model name and description and click **Save**. Workflow Designer opens.



If you prefer, you can edit an existing model by selecting a model from the Process Models page and clicking **Edit**. In this case, the page shows:

- The detailed summary and the selected process model in read-only mode.
- The current active version number of the model.
- The model description and task details.
- History of the model and the ability to select the history version.

To edit the selected model, click **Visual Editor** , which opens the selected workflow in edit mode.



Note: Each time you edit a Business Process Model, you create a new version of it. You do **not** overwrite the original Process Model.

Step 2: Configure process parameters or variables


Process parameters are attributes that do not have a predefined value or do not come from the database. A **process parameter** affects the entire Business Process, for example, Name or Location. To start a Business Process, a person may have to select a parameter set or change values.

1. Click in an empty area of the process model.
2. Edit the parameters in the properties panel.

Process variables add data globally to a process run. You can define and initialize variables that have not already been initialized in reference forms, [script tasks](#), and [public REST APIs](#). To do this, expand the **Activities** group, select **Initialize Variables**, drag it to the workspace, and attach it to a process. For more information, refer to [Initialize Variables task on page 59](#).

Note:

- The **Initialize Variables** activity is used to initialize variables when the Business Process is initialized regardless of where or how many times the activity appears in the flow.
- The **Initialize Variables** activity does not overwrite the value of any variables that are initialized as part of the call to the Business Process (for example, when an API is used to launch the Business Process).

- a. Click in an empty area of the Process Model.
- b. In the **Variables** window, click Add  to create a process variable. Then enter a **Variable Name** and **Variable Type** that corresponds to the type of values that are expected.

Examples:


Variable Name = `EmployeeName`; Variable Type = `String`.

Variable Name = `Age`; Variable Type = `Integer`.

- c. Select a process variable. Click Delete .

Example: You can delete variables that are blank.

- d. Click **Save**.

 **Note:** The following notation is required to refer to a process variable: `${varName}`. A script task can also declare process variables, for example, `execution.setVariable("varDays, noOfDays+1);`


Step 3: Configure task parameters

A parameter is an attribute that does not have a predefined value or does not come from the database. A **task parameter** affects only a task within the process, and overrides the corresponding process parameter or default value.

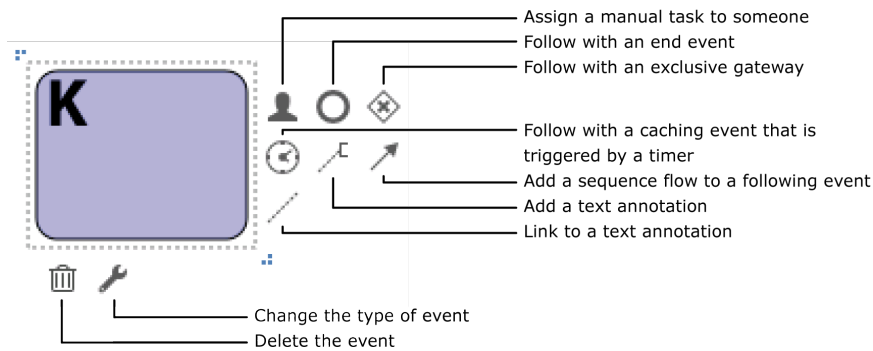
1. Select the task in the workspace.
2. Edit the parameters in the properties panel to configure the form.

Step 4: Link the events

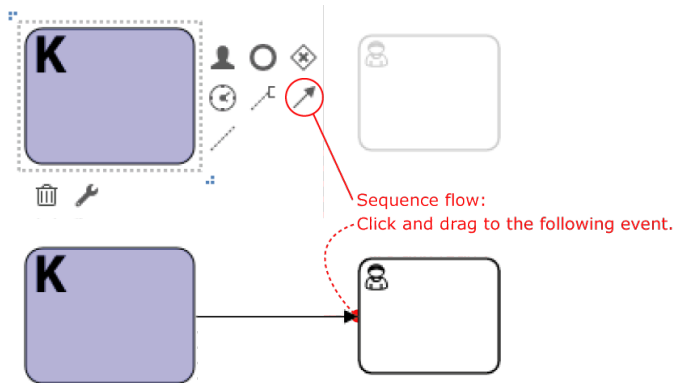
To move the process forward, complete the following steps:

 **Note:** To see detailed information about an event, hover the cursor over the event in the workspace.



1. Hover the cursor over an event to select icons.




2. From the first event, click and drag the Sequence Flow icon  to the following event.



Step 5: Finish and verify the process


1. End the process with an end event .
2. Click **Validate the Model** . Correct any errors.




Note: Models that fail validation cannot be deployed.


3. Click **Save the Model** .
 - Verify the settings.
 - Click **Save and Close Editor** or **Save**.


Step 6: Deploy the Model

Deploy Business Process Models to install and make them available for use. Also deploy models to test modifications.

Note: If you cannot deploy a model, it may be invalid. Open the model in the Process Designer, click **Validate the Model** , and correct any errors.

1. From the Main Menu, navigate to **Administration > Application Setup > Business Process Setup** and select **Process Models**.
2. From the Process Models page, select the applicable Process Model.
3. Click **Deploy** .
4. Complete the fields on the **Business Process** page.
 - a. Select one or more **Template Categories**, for example, Notification Templates, Overtime Approval Timer, External Actions, Request Transitions, and so forth.
 - b. In **Start Effective**, select the effective start date and in **End Effective**, select an end date or **Forever** to make the Business Process available indefinitely.
 - c. In **Status**, select **Active**.
 - d. In **Action List**, select whether to **Show** or **Hide** the Business Process.
 - e. In **Tile List**, select **Show** to enable the Business Process to be listed in the My Business Processes tile or **Hide** so it is not listed.
 - f. In **GoTo List**, select **Show** to enable the Business Process to be listed in the Go To links or **Hide** so it is not listed.
 - g. Click **Save**, then click **Return**.
 - h. The **Business Processes** page lists the deployed processes. Click **Refresh**.
5. Verify that the process is available.
 - a. Log in as the person who will run the process.
 - b. In Control Center, click **My Actions** . As an example, select **Business Processes > Manager Delegation**.
 - c. In **Delegate Profile**, select a person to take authority during the absence.
 - d. Select the **Start Date** and **End Date** from the calendars . The dates are in the time zone of the requester.

- e. Select the **Role Profile**. A role profile is associated with one function access profile (FAP) and one display profile to define the functions that a person can perform.
- f. Click **Submit** and close the panel.
- g. Click **Notifications**  and click **View All**.
- h. Check the **Tasks** in the Control Center or My Notifications.

 **Note:** Depending on how the workflow is configured, you can also verify that it is available by running it from the user's My Business Processes tile on the home page or from the GoTo control on the Timecard, Schedule, Leave of Absence Case Editor, Employee Search, People Information, or Dataviews.

Business Process Best Practices

Familiarize yourself with the following guidelines before creating Business Processes.

Overall

- Provide Assignment details in user tasks.
- Attach a Reference form to a user task if the Business Process has to be executed from Control Center.
- The model should pass all validation errors to be deployed successfully. Always validate your model after designing by clicking the **Validation** button on the editor. A common example omission is not providing the end event in the process definition. To validate your process model:
 1. After the model is created, click the **Validation** button on the editor.
 2. If there is an error, a red cross appears on top of the element.
 3. Click the red cross to open a popup window that contains applicable error messages to help you correct the error.
 4. When all validations are successful, a message appears on top of the model design "Process Model contains no validation errors."
- Every model should have unique Process Identifier value. By default, this value is created on the basis of model name provided.

If the Process Identifier value of the newly created model matches any of the already deployed models value, the new model cannot be deployed. It will throw an error while deploying on the Process Model setup page. You must manually update the value to a unique value.



- Execution variables are used to store data when a process is executed. Stored data is specific to a running process instance and cannot be shared with other process instances. Note the following:
 - **name_ column** carries the variable name. It has limit of 255 characters.
 - **text_ column** carries variable value data. It has limit of 4000 characters and the value of any execution variable cannot exceed this limit.
 - Variable values are not stored as encrypted. Therefore, any data that can possibly breach security should not be stored as an execution variable.
 - Use variables according to these best practices and avoid redundancy.
- By default, the frequency of the async/timer job cannot be less than 1 minute, even though these tasks can be defined in the workflow design with a smaller frequency. As a best practice, avoid any high load/volume WFM REST API invocations to be part of a shorter frequency timer jobs.

Design

- Put validations early in a Business Process so that you avoid later or redundant processing. For example, Get data from forms or web services at the start of a process.
- Prevent frozen processes. Always provide a path out from every decision point.
- Assign a form to the Start event so a cancellation stops the process and it does not remain open indefinitely. Submission of the form moves the process onward.

Diagrams

- Order the flow from left to right.
- Lay out sequence flows horizontally.
- Events in diagrams do not have to be oriented chronologically.
- Lay out data associations and message flows vertically.
- Make the primary or successful path dominant.
- Diagrams can include loops.
- Try to fit the diagram on a single page:
 - Use subprocesses to hide complexity that is secondary to the main flow, or to split a complex process into phases.
 - Use Call activities to reuse other processes.

- Depending on the intended user, create two versions of the diagram:
 - A summary diagram with all subprocesses and call activities collapsed, and data objects hidden
 - A verbose diagram that shows everything fully expanded
- Try to keep the paths simple and aligned neatly by using the following tools:
 -  Add a bend point.
 -  Remove a bend point.

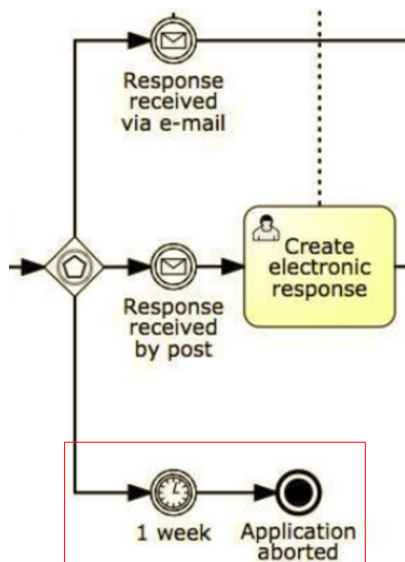
Gateways

Gateways determine what path is taken through a process that controls the flow of both diverging and converging sequence flows. A single gateway can have multiple inputs and multiple output flows. The term “gateway” implies that there is a gating mechanism that either allows or disallows passage through the gateway. In other words, as tokens arrive at a gateway, they can be merged on input and/or split apart on output as the gateway mechanisms are invoked. If the flow does not need to be controlled, then a gateway is not needed.

Best practices are:

- Avoid flows that cross other flows.
- Do not mix diverging with converging gateways.
- Put an activity that states the decision conditions just before a diverging gateway. This allows you to interrupt this decision activity if needed.

Example of a timer path added to a gateway:



Naming

Annotate everything. Use descriptive, clear labels for events, gateways, and sequence flows so that others can understand the elements of your design.

- Name all activities, events, and data objects.
- Avoid articles and pronouns.
- For process steps, use a verb plus noun structure, for example: **Check availability**.
- For input steps, use a noun plus verb structure, for example: **Request received**.
- Label diverging exclusive gateways (XOR gateways) used to create alternative paths within a process flow as a question, for example: **True?** and **False?**
- Use text annotations when a step, gateway, or group is not obvious.

A diverging Exclusive Gateway (or XOR Gateway) is used to create alternative paths within a Process flow. For a given instance of the Process, only one of the paths can be taken. When the execution of a workflow arrives at this gateway, all outgoing sequence flows are evaluated in the order in which they are defined.

Chapter 2: Start Events

A start event indicates where a process starts. The type of start event defines how the process is started, for example, the process could start on arrival of a message or at specific time intervals.

Start events are always "catching," which means that the event is waiting until a certain trigger happens.

○ None start event

A *None start event* means that the trigger for starting the process is unspecified and that the engine cannot anticipate when the process instance must be started.

A None start event can have a start form that is displayed when selecting the process definition from the processes list. Note that a process instance is not started until the start form is submitted. A None start event without a form simply has a button displayed to start the process instance.

Note: A subprocess always has a None start event.

A None start event has the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Execution listeners	Execution listeners configured for this element instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Process Initiator	The process variable in which the user ID of the initiator of this instance should be stored.
Form key	A key that provides a reference to a form. This property is available for compatibility with the Alfresco Process

Property	Description
	Services community, but should not be used directly when using Forms. Use the Referenced form property instead.
Referenced form	A form reference.
Form properties	A form definition with a list of form properties. Form properties are the way forms are defined in the community version of Alfresco Process Services. Configuring them has no impact on the rendered form in the Alfresco Process Services, the Referenced form property should be used instead.

⊕ Start timer event

A *Start timer event* starts a process or subprocess at a specific time. You can use it for processes that start only once and for processes that start in repeated time intervals. A Start timer event is scheduled as soon as a process is deployed.

A process instance started by a Start timer event cannot have a start form, because it is started by the system. Similarly, it does not have a process initiator like a None start event. Because of this, the assignment "assigned to process initiator" does not work.

Note: A subprocess cannot include a Start timer event.

A Start time event has the following properties:

Property	Description
ID	A unique identifier for this instance.
Name	A name for this element.
Documentation	A description of this element.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Time Cycle	A timer cycle defined in http://en.wikipedia.org/wiki/ISO_8601 [8] format, for example: R3/PT10H.
Time Date in ISO-8601	A point in time defined as a http://en.wikipedia.org/wiki/ISO_8601 [8] date, for example: 2015-04-12T20:20:32Z.
Time Duration	A period of time defined as a http://en.wikipedia.org/wiki/ISO_8601 [8] duration, for example:PT5M.

△ Start signal event

A *Start signal event* is an event that references a named signal. A signal is an event of global scope and is delivered to all active handlers (waiting process instances/catching signal events). The signal is fired from a process instance using the [intermediary signal throw event](#) or programmatically through the Java or REST API. In both cases, a process instance for any process definitions that have a Start signal event with the same name are started. You can select a synchronous or asynchronous start of the process instances.

A Start signal event has the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Signal reference	The name of the signal that initiates this event. Note that signal references are configured on the root level of the process instance and then linked to the signal start event via this property. To configure it, deselect any other element and click the Signal definitions property.

Ⓜ Start error event

A *Start error event* catches an error from an error boundary or error end event and starts an event subprocess. A Start error event cannot be used for starting a process instance.

Start error events have the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Error reference	The name of the error that initiates this event. This reference needs to match the error identifier thrown by the event that throws the particular error.

Chapter 3: Activities

Activities complete a task within a process. Users must complete the tasks according to the design and flow of the process. The flow sets the sequence from task to task. The entire process is called the root task.

The following types of tasks are available:

- [User tasks on page 29](#)
- [Script tasks on page 32](#)
- [Mail tasks on page 38](#)
- [Decision tasks on page 40](#)
- [API service tasks on page 44](#)
- [Generic Rest Connector on page 48](#)
- [Internal REST Connector on page 55](#)
- [Initialize Variables task on page 59](#)
- [Attestation Text Substitution on page 67](#)

User tasks

As its name implies, a *user task* is used to model work that needs to be done by a person. When the process execution arrives at a user task, a new task is created in the task list of the users or groups assigned to that task.

 **Examples:** Request time off, request leave, approve or refuse a request.

Types of user tasks include:

- **Initial task:** A person initiates a Business Process from the Control Center. If that person cancels the task, the process stops before it starts.

- **Assigned task:** A person receives a notification to complete a task. If that person cancels the task, the notification and the process remain active.

User tasks can be assigned to a:

- **Task Owner** – The person who completes the task
- **Task Assignee** – The person who is assigned to complete the task

Most of the time, one person is both the task owner and the task assignee. However, a task owner can delegate a task to someone else. For example, a manager (task owner) delegates a task to an assistant (task assignee).

Assignments can be made as an **Explicit Assignment** from the Process Model, a variable, service call, a delegate expression, or by delegation.

- **Candidate Assignment** – To individuals or groups

Individuals who are candidates or who are a member of a candidate group can claim a task.

User tasks have the following properties:

Property	Description
ID	A unique identifier for this element
Name	A name for this element.
Documentation	A description of this element.
Assignment	Configures to whom this task should be assigned. It is possible to use Fixed Values (advanced usage: these are Alfresco Process Services expressions, for example by invoking a class or Spring bean) or use the Identity Store option. It is recommended to use Identity Store to automatically assign to Process initiator : <ul style="list-style-type: none"> • Assigned to process initiator. • The user that started the process instance will be the assignee of this task. • Assigned to single user • A single user who will be the assignee of the task. This user will see the task in their Involved tasks task list. It is possible to reference a user that was selected in a previous form field (tab Form field).
Referenced form	Allows you to configure or create the form for this task. This form (also called a task form) is rendered when the task is shown in the task list of the user. A user task typically has a form defined.
Form key	This is a property that exists for compatibility with the community version. When working with task lists and forms, do not set this property.
Form properties	This is a property that exists for compatibility with Alfresco Process Services community. When using Alfresco Process Services to work with task lists and forms, do not set this property.
Due date	Allows you to configure a due date for the task. In the task list, tasks can be sorted by due date to see which tasks

Property	Description
	<p>are needed to be completed the soonest. The possible ways to configure this property are:</p> <ul style="list-style-type: none"> • No due date – This is the default value. • Expression definition (Advanced) Uses an Alfresco Process Services expression to resolve the due date (for example, this expression could call a Spring bean). • Fixed duration after task creation – Allows you to configure an amount of time, starting from the creation of the task. • Based on field – Allows you to configure the due date based on a previous field in the process instance, by adding or subtracting a certain amount of time. • Based on variable – Allows you to configure the due date based on a variable previously declared in the process instance, by adding or subtracting a certain amount of time.
Allow email notifications	When enabled, an email is sent to the assignee when the task is created.
Asynchronous	(Advanced) Define this task as asynchronous. This means that the task will not be created as part of the current action of the user and will be created later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This task is useful to solve race conditions.
Execution listeners	Execution listeners configured for this instance. An execution listener lets you execute Java code or evaluate an expression when an event occurs during process execution.
Multi-Instance type	<p>Determines if this task is performed multiple times and how. The possible values are:</p> <ul style="list-style-type: none"> • 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 the previous one.
Cardinality (Multi-instance)	The number of times the task is to be performed.
Collection (Multi-instance)	(Used with Multi-Instance type) The name of a process variable that is a collection. For each item in the collection, an instance of this task will be created.
Element variable (Multi-instance)	A process variable name 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. You can specify an expression here to be evaluated each time an instance ends. If the expression evaluates to true, all remaining instances are destroyed and the multi-instance activity ends.
Used for compensation	If this activity is used for compensating the effects of another activity, you can declare it to be a compensation handler.

For information about using user tasks in forms, refer to [Create Forms on page 109](#).

Script tasks

A script task is an automatic activity. When a process execution arrives at the script task, the corresponding script is executed. A script task executes a script to directly access Java beans in the configuration as well as task and execution beans. Script tasks achieve most of the functions of Java classes.

Examples of script tasks include:

- Initialize variables at the start of a process.

Tip: Use a script task immediately after the start event to initialize variable values or retrieve data from a web service.

- Parse form fields.
- Access exposed spring bean properties.
- Make decisions based on simple logic.
- Transform data.

Note:

Workflow Designer includes some built-in protection for scripting, but be aware of the following:


- Make sure that you are not ending up in an infinite loop.
- Do not create large in-memory objects that can consume too much memory.
- Do not use system level packages such as `java.lang.System` that can affect your system's health.

Script tasks include the following properties:

Property	Description
Script format	The JSR-223 [16] name of the scripting engine your script is written for. By default, Alfresco ProcessServices supports javascript and groovy formats.
Script	The script that will be executed.
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Variables	In the script, it is possible to set new process variables (using <code>execution.setVariable(myVariable, myValue)</code>),

Property	Description
	however these do not display automatically in drop-downs later on (like the sequence flow condition builder, forms, etc.). Configure this property with the variables that are set or exported by this script task.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Asynchronous	(Advanced) Define this task as asynchronous. This means that the task will not be executed as part of the current action of the user, but later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Multi-Instance type	Determines if this task is performed multiple times and how. The possible values are: <ul style="list-style-type: none"> • 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)	The number of times the task is to be performed.
Collection (Multi-instance)	The name of a process variable that is a collection. For each item in the collection, an instance of this task is created.
Element variable (Multi-instance)	A process variable name that 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. You can specify an expression here to be evaluated each time an instance ends. If the expression evaluates to true, all remaining instances are destroyed and the multi-instance activity ends.
Is for compensation	If this activity is used for compensating the effects of another activity, you can declare it to be a compensation handler.

Languages for script tasks

 **Caution:** Scripts are a powerful way to accomplish tasks or they can do a lot of damage. You **must** be qualified in the script language before you use a script task.

- **Groovy** is strongly recommended because of fast performance, and it includes JsonSlurper to parse JSON responses from APIs.

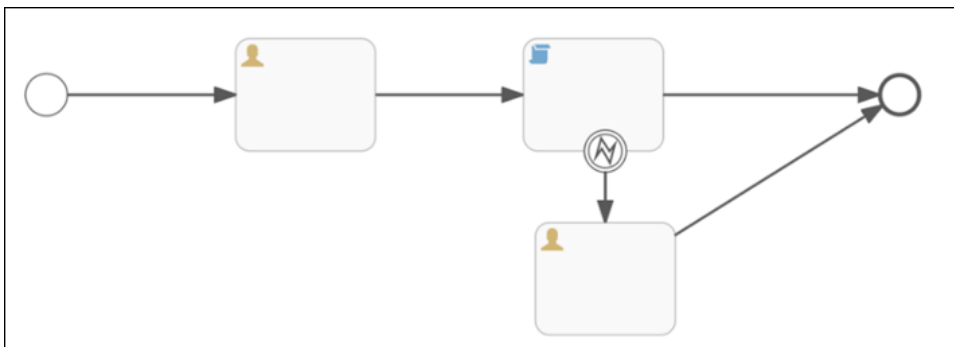
- **JavaScript** is supported but it suffers from slow performance, is less flexible, and makes it hard to access Java functions.
- Any JSR-223 compatible language can be used. Java Specification Requests (JSR-223) defines a framework for embedding scripts into Java source code.

A script task can declare process variables: `execution.setVariable("varDays", noOfDays+1);`

To refer to a process variable or other expression from a text field, use the following notation: `${varName}`

Error handling in script tasks

If the script generates a runtime exception, an error message is displayed. To minimize the chance of breaking a process due to an error, you can use a boundary event to move the flow to another route.



For example,

```

try{
    undefin();
}catch(error){
    execution.setVariable("error_message", err.message);
    throw new org.activiti.engine.delegate.BpmnError ("error_
message",err.message) '
}
  
```

Script task examples

The following are some commonly used script tasks:

- [Start a process from a timer on page 35](#)
- [Create an initialization script on page 35](#)

- [Get values from form fields on page 36](#)
- [Example: Reminder Process Model on page 37](#)
- [Validation script on page 38](#)

Start a process from a timer

To start a Business Process from a timer, you can use a script task with the following properties:

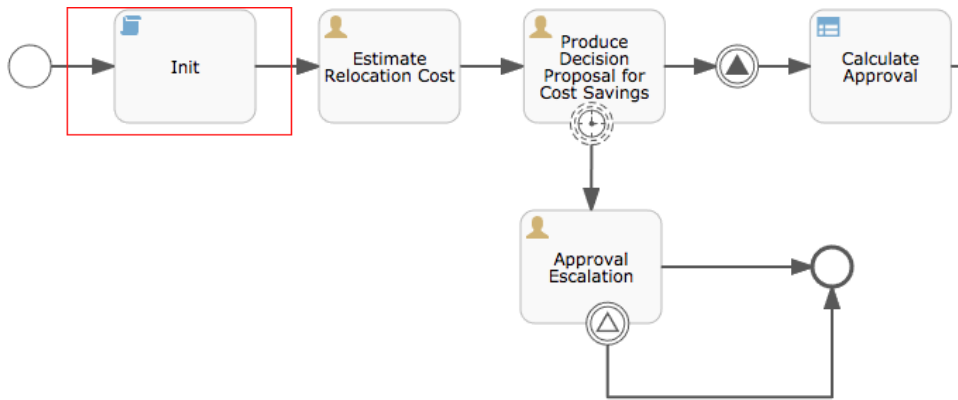
- `execution.setVariable("ext_tenantId", "manufacturing");`
- `execution.setVariable("initiator", "10015");`

Note: The `wfc_url` parameter is optional and not needed to initialize the script task.

Create an initialization script

You can start a Business Process with an initialization script task that automatically imports data – such as person or group information – for use throughout the process. This can improve performance by reducing the number of times to import data. You can add a timer start event just before the script task to start the process at a specific time. Or an initial user task could require a person to enter information in a form from the Control Center to start the process.

1. Insert a **Script Task** immediately after the start event. for example:



2. Select the script task. In the Properties section:
 - a. Enter a **Name**, for example: `Init` or `Initialize`
 - b. In **Script Format**, enter `groovy`.

- c. Select **Script**.
- d. Enter an array of user or group IDs.

Example initialization of an assigneeList process variable

```
import com.activiti.domain.idm.User;
import com.activiti.security.SecurityUtils;
out.println('Start - Initialize moving estimate');
User currentUser = SecurityUtils.getCurrentUserObject();
ArrayList idList = new ArrayList();
Long tenantId = currentUser.getTenantId();
List<User> users = userService.getAllUsers(0,999,tenantId);
for (User u : users) {
    idList.add(u.getId());
}
execution.setVariable('assigneeList', idList);
out.println('End - Initialize moving estimate');
```

3. Click **Save**.

Get values from form fields

You can access form field values by way of a script task with the following properties:

```
var varName = execution.getVariable("FormFieldID");
```

where:

- FormFieldID is the ID associated with that form field.
- The value of this field is the value of the form field.
- In the case of radio buttons and drop-down selection lists, the value is the ID of the selected button or value. For example, you can get the selected value of a radio button.

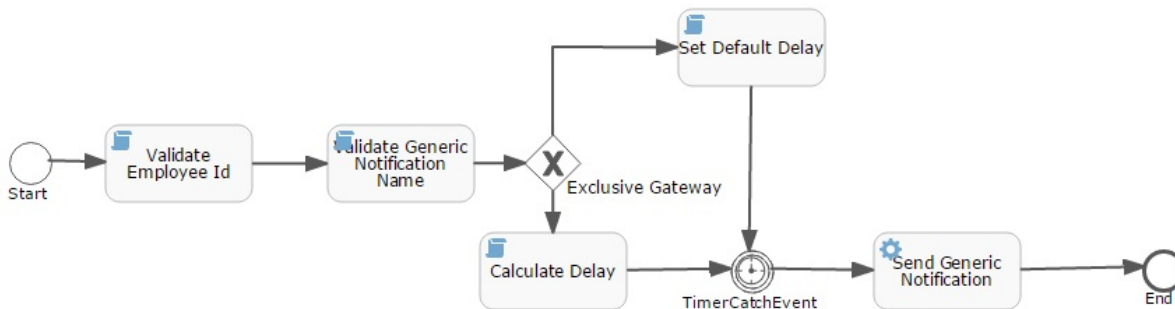
To get access to a variable that is created in a script outside the script task, you must set it as an execution variable in the script as follows:

```
execution.setVariable("varName", "varValue");
```

Execution variables are accessible outside of scripts, for example in service task properties and forms, as follows:

```
${executionVariableName}
```

Example: Reminder Process Model



This preconfigured Business Process Model sends reminder notifications.

- **Validate Employee ID** – Verify that the employee identification number is specified; if not, send an error message.


```

var EmployeeID = execution.getVariable("EmployeeID");
if(EmployeeID == null || EmployeeID.isEmpty() ){
    execution.setVariable("ErrorString", "Missing EmployeeID");
}
  
```

- **Validate Generic Notification Name**
 - Verify that the notification is specified; if not, send an error message.
 - Similar script to Validate Employee ID
- **Exclusive Gateway**
 - If the notification timer `${GenNotificationTimerDuration == ""}` is specified, calculate the delay.
 - If not, use the default date based on GenNotificationDate.
- **TimerCatchEvent** – Set the delay (**Calculate Delay** or **Set Default Delay**) before continuing the process.
- **Send Generic Notification** – Send the notification. The execution variables are defined in the Generic

Notification API Service Task and provide the input parameters for the notification.

Name	Implementation
resourceId	\${initiator}
personNumber	\${EmployeeID}
emailList	\${GenNotificationEmailList}
tagList	\${GenNotificationTAGlist}
names	\${GenericNotificationName}
recipientList	\${GenNotificationRecipientList}



Validation script

```
requestItemsList = execution.getVariable("RequestedItems");
execution.setVariable("CUSTOM_VALIDATE_RESULT", "1");
var hasReqOn = execution.getVariable("hasRequestOn");
var canSubmitRequest = execution.getVariable("CanSubmitRequestForApproved");
if(canSubmitRequest == "false"){
    execution.setVariable("showErrorMsg", "true");
}
var showErrorMsg = execution.getVariable("showErrorMsg");
if(hasReqOn == "true" || showErrorMsg == "true"){
    execution.setVariable("CUSTOM_VALIDATE_RESULT", "0");
    execution.setVariable("CUSTOM_VALIDATE_ERROR_MESSAGE", "The request conflicts
with a previously approved request. Verify the information in the request and submit
again.");
}
```

Mail tasks

You can enhance Business Processes with automatic mail service tasks that send e-mails to one or more recipients. This task requires an SMTP server.

Mail tasks have the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
To	The recipient of the e-mail. You can specify multiple recipients in a comma-separated list. When using a fixed value, this can be an expression. Like with the user task, it is also possible to use the Identity store option here to pick users that are known in the system or to reference people that were selected in form fields prior to this email task.
From	The sender's email address. If you do not specify this, the default configured system-wide setting from address is used. This can be an expression.
Subject	The subject of this email. This can be an expression.
Cc	The cc list for this email. You can specify multiple recipients in a comma-separated list. This can be an expression.
Bcc	The bcc list for this email. You can specify multiple recipients in a comma-separated list. This can be an expression.
Text	The text content of this email. You can specify this as well as HTML to support email clients that do not support rich content. The client will fall back to this text-only alternative.
Html	The HTML content of this email.
Charset	The charset for this email. By default UTF8 will be used.
Asynchronous	(Advanced) Define this task as asynchronous. This means the task is not executed as part of the current action of the user, but later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Multi-Instance type	Determines if this task is performed multiple times and how. The possible values are: <ul style="list-style-type: none"> • 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)	The number of times the task is to be performed.

Property	Description
Collection (Multi-instance)	The name of a process variable that is a collection. For each item in the collection, an instance of this task is created.
Element variable (Multi-instance)	A process variable name that 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. You can specify an expression here to be evaluated each time an instance ends. If the expression evaluates to true, all remaining instances are destroyed and the multi-instance activity ends.
Is for compensation	If this activity is used for compensating the effects of another activity, you can declare it to be a compensation handler.

Decision tasks

A Decision task is used to select a decision table while designing your process model. A decision table enables you to define a set of business rules that are applied when it is executed.

Decision tasks have the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Reference decision table	Defines the actual decision table to be executed. The decision table can be part of the process definition (a so-called embedded decision table) or defined on itself (a so-called reusable decision table).
Asynchronous	(Advanced) Define this task as asynchronous. This means the task will not be executed as part of the current action of the user, but later. This can be useful if it's not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Multi-Instance type	Determines if this task is performed multiple times and how. 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.

Property	Description
Cardinality (Multi-instance)	The number of times the task is to be performed.
Collection (Multi-instance)	The name of a process variable that is a collection. For each item in the collection, an instance of this task is created.
Element variable (Multi-instance)	A process variable name that 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. You can specify an expression here to be evaluated each time an instance ends. If the expression evaluates to true, all remaining instances are destroyed and the multi-instance activity ends.
Is for compensation	If this activity is used for compensating the effects of another activity, you can declare it to be a compensation handler.
Dynamic Decision Table (CHECKBOX)	Using this switch, the user can define a decision table that does not ask for redeploying the process model every time the decision table is updated. It adds the ability to update the decision table and reflect the changes on published and existing Business Processes without redeploying the process models.

Decision tables

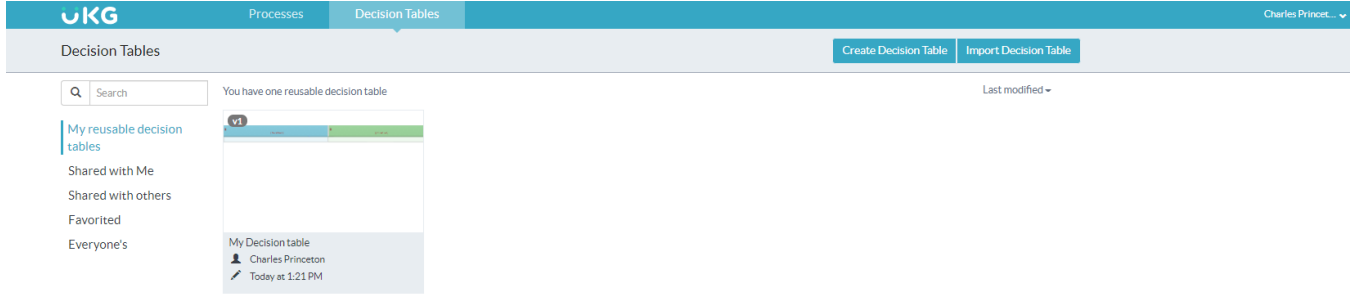
Decision tables enable the user to make a decision based on a set of conditions during the execution of a Business Process. The user can define input and output variable(s) and decide on their values on the basis of the rules defined in the table.

The decision table can be created/accessed through the Decision task in the stencil set as well as through the **Decision Table** tab in Workflow Designer. You can also import, export, delete, and copy these tables from the **Decision Table** tab.

When creating a decision table, you define input and output variable(s) with their output values based on the rules defined in the table.

- **Input variables** – Used to define the condition(s)
- **Output variables** – The results of the condition(s) defined.

The Business Process model uses these variables to populate values dynamically during the execution of the Business Process.



The decision table workspace displays:

- The current active version number of the table.
- Decision table description and details.
- History of the table and ability to select history version.

From the workspace, you can:

- Edit, copy, and delete the decision table
- Mark the table as a favorite.

You can also export a decision table so it can later be imported from for use with another model.

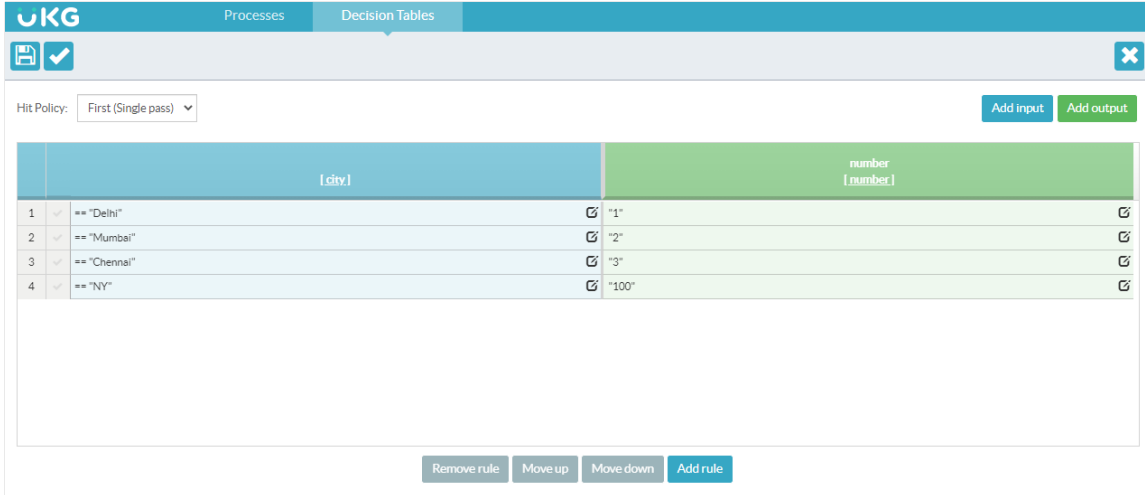
Decision table editor

To create a decision table:

1. Select **Activities** > **Decision task** from the Workflow Designer stencil set.
2. Select the **Referenced decision table** field at the bottom of the workspace.
3. Click **New Decision Table**.
4. Give the table a name and description, then click **Create Decision Table**.

To edit an existing decision table, click the **Decision Table** tab and select one.

Whether you are creating or editing a decision table, the decision table editor opens.



From the decision table editor, you can:

- Configure multiple input variables by clicking the **Add input** button.
- Configure multiple output variables by clicking the **Add output** button.
- Add multiple business rules by clicking the **Add rule** button.
- Define the order of the rules by clicking the **Move up** and **Move down** buttons.
- Delete the rule(s) by clicking the **Remove rule** button.
- Select the Hit Policy (Single Pass only)
 - **A(ny)**: Multiple rules can match; all must have the same output.
 - **F(irst)**: Select the first matching rule based on the order in the table.

Decision table example

⚠ Caution: The order of rules in the table dramatically determines the results.

AF (hit policy)	Cost of relocation (input)	Distance (input)	Manager approval (output)
1	<= \$500	<= 100 km	Not required
2	<= \$500	> 100 km	Required
3	> \$500	n/a	Required

Results

After a decision table runs, the process instance view shows detailed information about the run in the Calculated and Input values.

Limitations

When using decision tables, be aware of the following:

- Do not use the backslash (\) or double quote (") characters when defining the input or output variables. Workflow Designer treats the backslash (\) as an escape character and the double quote (") as the end of a string, not as text characters.
- When you import a decision table (.dmn file), the imported decision table does not display the label information for the input variable. In spite of this, the variable works correctly when the Business Process is executed.
- Edit the output variable in the imported decision table even if there are no changes and save it before creating a model. If you do not do this, the imported decision table variables do not populate the **Variable** drop-down of form fields when editing reference forms.

API service tasks

API service tasks interface with external systems where the application completes service tasks automatically, for example, "Get domain information."

The API service task uses the API to:

- Start a Business Process.
- Get and submit data from forms.
- Get information about process completion, execution variables, previous tasks.

The task obtains domain information as follows:

- Custom web service integrations
- Generic service task
- Generic REST service

REST API Calls

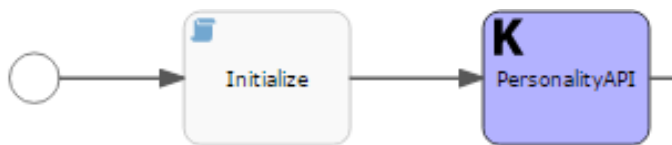
This task can call the following REST APIs:

- Personality API
- Delegate Authority API
- Delegate Profile API
- Delegator API
- Role Profile API
- Resume API
- Generic Notification Notify API
- iHub API
- Request API
- Request Status Change API
- Accruals API

⚠ Caution: By default, a Business Process retrieves data by using the credentials of the person who initiates the process. However, a process may need to retrieve data that the initiator of the process cannot access. Do this **only** when absolutely necessary and be careful to **not** expose data to people who are not privileged to access that data. To access this data, make the API request on behalf of another person; a best practice is to create a user account specifically for this purpose rather than use another real person's credentials.


Configure a ADP service task

1. From the palette, open **ADP APIs** and click and drag a service task to the relevant position in the process. For example:



2. Select the service task.
3. Provide the following information in the Properties panel at the bottom of the workspace:
 - a. Enter a **Namesuch** as `PersonalityAPI`.
 - b. Select **{APIName} Input/Output**, for example, `Personality API Input/Output`.
 - c. Enter the **Resource ID**, for example, `${initiator}`.

- d. Select the **API**, for example, `Person Information`.
- e. Select the **Actions** such as `Load`.
- f. Select or enter any other parameters that are required for the API.
- g. In **ID Source**, select **Current User**. Other examples are `Person Number`, `Badge Number`, `Employee Key`, `Person Key`, `User Key`.
- h. In **Input Parameters**, enter a text value, expression, or select a variable to define the value of each parameter.
- i. In **Output Parameters**, enter the **Variable Name** for each parameter. For example, enter **personData** for person data.
- j. Click **Save**.

 **Example:** To parse the results of the Personality API call, add a script task after the ADP service task to parse the JSON response and populate the `firstName` and `lastName` process variables.

```
import groovy.json.JsonSlurper;
import groovy.json.JsonOutput;

jsonSlurper = new JsonSlurper();


listRest = jsonSlurper.parseText(execu,on.getVariable
("personData"));

mapPerson = listRest.get(0);

personNumber = mapPerson.get("PersonNumber");

personName = mapPerson.get("firstName");

execution.setVariable("personName",personName);
```

4. Add the variables in the script task to the forms.
5. Click in a blank part of the workspace.
6. In **Properties**:
 - a. Select **Variables**.
 - b. Select **Add Variable** .
 - c. Enter a **Variable Name**. Examples: `firstName`, `lastName`.

d. Select a **Variable Type**.

string

integer

boolean

people

group

e. Click **Save**.

7. Select a user task. for example, `Estimate Relocation Cost`, then add or open the form for that task and add a display text:

a. Select **Referenced Form**.

b. Select **New Form** or Select the form and select **Open**.

c. Select or add the field. Click **Edit** .

d. Enter a **Name**. Example: `Greeting`


e. Enter a **Text to Display**. Example: `Hello`

f. Select the **Variable**. Example: `${personName}`
Example Text to display = `Hello ${personName}`

g. Click **Close**.

8. Save the form:

a. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.


b. Click **Save the Model** .

◦ Verify the settings.

◦ Click **Save and Close Editor** or **Save**.

9. Save the Process Model:

a. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

b. Click **Save the Model** .

- Verify the settings.
- Click **Save and Close Editor** or **Save**.

10. Deploy, run, and test the model.

Generic Rest Connector

A Generic Rest Connector is a custom REST connector that was introduced to enhance the Activiti Enterprise REST Connector. It is used to call Global Rest APIs. These Global REST API calls are made routing through a proxy server.

Generic Rest Connectors have the following properties:

Property	Description
Name	A name for this element.
Documentation	A description of this element.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
URL	Fully qualified endpoint URL = protocol + host url + rest path + query parameters
HTTP Method	GET/POST/PUT/DELETE
Secure Connection	Determines if http or https will be used. This overrides whatever is defined in endpoint URL.
Payload	The content of any PUT or POST. The content type is defined as application/json
Headers	A simple JSON array that includes any headers you may want added to the request (for example cookies). Format : [{"name":"<name of header>", "value":"<value of header>"}]
Response Variable	Name of the variable that the response will be mapped to.

The Generic Rest Connector calls any REST API by way of the API Gateway. This connector is **not** optimized for internal REST APIs.

You can use the Generic Rest Connector or the [Generic API Task](#) to call any API that does not have a service task. The Generic Rest Connector is the more flexible but technical method.

⚠ Caution: By default, a Business Process retrieves data by using the credentials of the person who initiates the process. However, a process may need to retrieve data that the initiator of the process cannot access. Do this **only** when absolutely necessary and be careful to **not** expose data to people who are not privileged to access that data. To access this data, make the API request on behalf of another person; a best practice is to create a user account specifically for this purpose rather than use another real person's credentials.

Example: Process with a Generic REST Connector



Configure a Generic Rest Connector

1. From the Stencil Set, select **Activities**, then click and drag a **Generic REST Connector** service task to the relevant position in the process.
2. Select the task.
3. In **Properties**:
 - a. Enter a **Name**, for example, `GetRequestData`
 - b. Enter the **HTTP Method**, for examples, `GET`, `POST`, or `PUT`
 - c. If the method is `POST`, enter the required **Payload** as a string.
 - d. Enter the **ResponseVariable** that stores the response from the REST API, for example, `restResponse`.
You can use this response later in the process as an execution variable:
`execution.getVariable("<variableName>")`
Also, you can use this variable to put values on forms, for example, `${<variableName>}`
 - e. Enter the **URL** for the REST API to call, for example, `http://localhost:8080/getData`.
Tip: If the domain of the URL matches the ADP domain, execution goes through the API gateway. Path and query parameters in the URL are retained, and **only** the host name and rest context are replaced with the API Gateway host and context.
 - f. Select **Secure Connection**.

- g. Enter the required **Headers** as strings.



Note:

If the ADP domain is found, the headers parameter is skipped.

Workflow Services handles authentication, so you do not need to provide cookies in headers for authentication.

(Optional) Provide authentication cookies in the header in a JSON string:

```
AUTHN_TOKEN, JWK_URI, IDP_URI, iPlanetDirectoryPro
```



Example script task to add cookies

```
headerVar =  
" [{"name\": \"Cookie\", \"value\": \"AUTHN_  
TOKENN="+execution.getVariable('AUTHN_TOKEN')+", JWK_  
URII="+execution.getVariable('JWK_URI')+", IDP_  
URI="+execution.getVariable('IDP_  
URI')+", iPlanetDirectoryPro="+execution.getVariable('TOKEN')+\""} ]";
```



Caution: AUTHN_TOKEN and other cookie values must be set in the execution variable.

4. Click **Save**.

Sample code for a generic REST connector

```
endpointUrl : http://ip.jsontest.com  
httpMethod : GET  
isSecure : false  
responseMapping : responseOutputVar
```

This REST API gets the IP address of the machine.

Other properties such as Payload must be left blank (default) because these are not required in this request.

The `responseOutputVar` can be used as `${responseOutputVar}` in a **Display Text** form to see the response in the Control Center.

Sample parameters to use as execution variables

- **RequestID** – Key to access the request ID for the suspending process
- **EmployeeID** – Key to access the employee ID for the suspending process

- **QueryDateSpan** – Date span of the request or a span that includes one of the requests
- **PersonNumber** – Person number of the requester
- **FlowLabel** – Flow label
- **RequestSubtype** – Request subtype
- **CurrentApproverID** – Key to access the employee ID for the suspending process

Generic Web Service API Task

This service task extends the [Generic API Task](#) and accepts any service task. In addition, the web service API task invokes a web service and uses the metadata to invoke the webservice API in the Business Process.

Mapping between user-interface fields and service-task class fields

–	UI label	Class field	UI element type	Values
API	API	choice	Drop-down	<ul style="list-style-type: none"> • The API to run • Comes from the corresponding request XSL File
Action	Action	choiceAction	Drop-down	The action to complete
Choice	Any depending on the choice	choice	Drop-down	The choice to use
ID source	ID Source	choiceIdSource	Drop-down	The choice identifier source to use
input-properties/output-properties	Any	In execution-class: <ul style="list-style-type: none"> • The mapping should be the same as defined in the POJO class. Plain old Java object (POJO) is an ordinary Java object that is not bound by special restrictions and does not require any class path. Example: <code>PersonalityAPITask.java</code> • Only for output-properties, use the execution-class value by removing the <code>output</code> keyword from the 	Text field	The input and output properties to use

–	UI label	Class field	UI element type	Values
		corresponding field in the POJO. Example: If the POJO contains <code>outputAction</code> , use <code>Action</code> as the execution-class.		

Configure a web service API task

Use the corresponding `GenericWebServiceAPITask.java` service task POJO to create the Generic Web Service task JSON file.

Name the JSON file **GenericWebServiceAPITask**. Unlike the [Generic Upload API Task](#), the generic web service task does not require a new service task that has new POJO and XSLT files.

Key elements

- **selects** – A JSON array that includes all choices. Key elements are name, label, and choices. A select can contain multiple choices.
- **choices** – A JSON array that includes all values in a choice. Elements are name, label, input-properties, and output-properties. A choices can contain selects to include more values.
- **input-properties** – A JSON array that includes all required input parameters. Elements are name, type, and execution-class. The value of execution-class should be same as that of the input property in the POJO class.
- **output-properties** – A JSON array that includes all required output parameters. Elements are name, type, and execution-class. The value of execution-class should be same as that of the output property in the POJO class.

If output-properties uses execution-class, remove the `output` keyword from the POJO field. Example: If the POJO contains `outputAction`, use `Action` as the execution-class.

- **label** – The text of any field in the user interface.
- **name** – Maps the field with the corresponding XSLT files. Also used as the `activiti-field` name when exported as a template to the `bpmn.xml` file. If a field does not have a label, name can be used.
- **execution-class** – Maps an input/output parameter with the respective POJO classes.
- **type** – The type of field, for example, String.

webServiceConfiguration

In choices is the `webServiceConfiguration` key field (required) that defines all web service configuration mappings and details. This field is not visible in the workspace but is used internally. To see the configuration details, complete the Business Process template and open the exported BPMN XML file.

The details include:

- **wsConfigName** – The name of the web service configuration. Use the same name as in the web service configuration.
- **operationName** – The operation name to initialize the SOAP request.
- **portName** – The port name to initialize the SOAP request.
- **SOAPRequest** – The SOAP request where input parameters are used. The input-properties values must be used inside SOAPRequest, so include the values in SOAPRequest and put the input variable name within parentheses { }.
`<input-property name="{cityName}" value="{cityName}" />`
- **SOAPResponse** – The SOAP response that includes:
 - **ResponseName**: The name of the response
 - **ResponseAttrName**: The attribute name
 - **ResponseXPath**: The XPath to the response

JSON Structure Example

The input-properties (`countryName` and `cityName`) in SOAPRequest use parenthesis as follows:

- `<web:CityName>${cityName}</web:CityName>`
- `<web:CountryName>${countryName}</web:CountryName>`

The value of input-properties comes from the Business Process Models workspace.

```
{
  "selects": [
    {
      "label": "API",
      "name": "choice",
      "choices": [
        {
          "label": "Person Information",
          "name": "Person Information",
          "input-properties": [
            {
              "name": "cityName",
              "value": "${cityName}"
            },
            {
              "name": "countryName",
              "value": "${countryName}"
            }
          ]
        }
      ]
    }
  ]
}
```

```

"-type": "String",
"name": "countryName",
"execution-class": "countryName"
},
{
"-type": "String",
"name": "cityName",
"execution-class": "cityName"
}
],
"output-properties": [
{
"execution-class": "outputParams",
"type": "String"
}
],
"webServiceConfiguration": {
"wsConfigName": "ExchangeRate",
"operationName": "GetWeather",
"portName": "GlobalWeatherSoap",
"SOAPRequest": "<soapenv:Envelope
xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:web='http://www.webserviceX.NET'><soapenv:Header/><soapenv:Body><web:GetWeather><
web:CityName>${cityName}</web:CityName><web:CountryName>${countryName}</web:CountryName
></web:GetWeather></soapenv:Body></soapenv:Envelope${cityName}${countryName}>",
"SOAPResponse": [
{
"ResponseName": "WeatherResponse",
"ResponseAttrName": "GetWeatherResult",
"ResponseXPath": "//GetWeatherResponse/GetWeatherResult"
}
]
}
]
}

```

Internal REST Connector

The Internal REST Connector is a custom REST connector that was introduced to enhance the Activiti Enterprise REST Connector. It is used to execute internal REST APIs. These properties are available:

Property	Description
Name	A name for this element.
Documentation	A description of this element.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Rest Path	Rest end point (Example : /legacy_api/v1/Delegate/GetDelegatesByPersonNumber?personNumber=10015)
HTTP Method	GET/POST/PUT/DELETE
Payload	The content of any PUT or POST. The content type is defined as application/json.
Resource ID	The person number or username for the user on whose behalf the API is being executed.
Resource ID Type	The type of the Resource ID, which can accept either PERSON_NUMBER or USER_NAME.
ResponseVariable	Name of the variable that the response will be mapped to.

Note: URL encoding is not provided by the Internal REST Connector. The user must use Script tasks to encode URLs before using them in the Internal REST Connector to support their use cases. Consider the following sample script used to encode a locale policy:

```
def localeName=URLEncoder.encode("American English", "UTF-8");  
  
execution.setVariable("apiURI", "/v1/commons/locale_  
policies?name="+localeName)
```

The Internal REST Connector uses the API Gateway to call any internal REST API; API Gateway configurations are specified in the **kronos-custom.properties** file located at server path `/usr/local/kronos/tomcat/conf`. The Internal REST connector has these features:

- Compared to the [Generic Rest Connector on page 48](#), the Internal REST Connector is simplified and manages authentication based on the configuration of Resource ID and Resource ID Type properties:

- By default both properties show 'No value'.
- The Resource ID is a text field and the user can provide either a hardcoded value or a variable (`${variable}`), for example).
If the user specifies the Resource ID in the field, then the Resource ID is passed as the onbehalf user to retrieve cookies to execute the API. The configured Resource ID property does not modify the existing value of the resourceId execution variable.

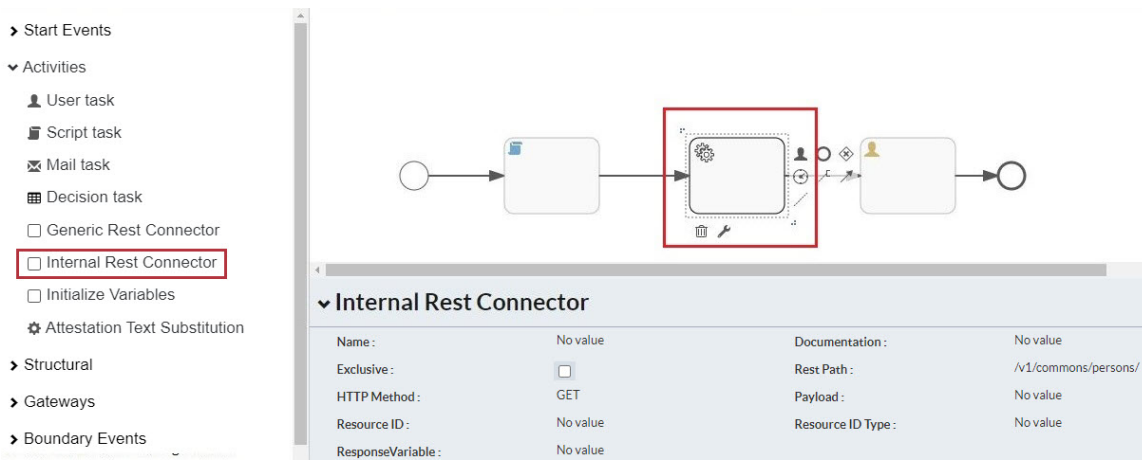
If the user does not specify the Resource ID, then the value present in the process execution variable is passed as the onbehalf user to retrieve cookies to execute the API.

- The Resource ID Type is a text field and the user can provide either a hardcoded value or a variable (`${variable}`), for example). The valid values are PERSON_NUMBER and USER_NAME. The system ignores invalid values and treats PERSON_NUMBER as the default. If the user specifies the Resource ID Type in the field, then that is what the system uses. The configured Resource ID Type property does not modify the existing value of the resourceIdType execution variable.

If the user does not specify the Resource ID Type, then the system uses the value present in the process execution variable. If it is not present in the process execution variable, the default value "PERSON_NUMBER" is used.

- The Internal REST Connector does **not** need:
 - A configuration file.
 - A tenant URL or REST context. Instead, you define a **REST Path**. Workflow Services handles the protocol, host name, and REST context automatically.
 - An App Key.
 - An authentication token. Workflow Services handles the tenant information and Rest API authentication automatically.
 - Any REST headers.

Example process with an Internal REST connector



Configure an Internal REST Connector

1. From the Stencil Set, select **Activities** , then click and drag a **Internal Rest Connector** service task to the relevant position in the process.
2. Select the task.
3. In **Properties**:
 - a. Enter a **Name**, for example: `GetRequestData`.
 - b. Enter the **Rest Path** for the API.
Example: `/v1/commons/persons/`
The path can include query parameters.
 - c. Enter the **HTTP Method**, for example, GET, POST, or PUT
If the method is POST, enter the required **Payload** as a string.
 - d. Enter a hardcoded or variable value for the **Resource ID**.
 - e. Enter a hardcoded or variable value for the **Resource ID Type**. Valid values are PERSON_NUMBER and USER_NAME.
 - f. Enter the **ResponseVariable** that stores the response from the REST API, for example, `restResponse`.
You can use this response later in the process as an execution variable:

```
execution.getVariable("<variableName>")
```

You can also use this variable to put values on forms, for example, `${<variableName>}`.

4. Click **Save**.

Initialize Variables task

A process instance and user tasks can include *process variables*, which are visible throughout the process, and *initialization variables*, which are specific pointers to where the process is active. A process instance can have any number of variables. You use the Initialize Variables task to define and initialize variables that have not already been initialized in [forms](#), [script tasks](#), and [public REST APIs](#). These variables can be used across various steps within a process model using the standard BPMN notation for expressions: **\${varName}**.

After you select and attach the Initialize Variables task in a process flow, you click the configuration link **Initialize Variable** to open the **Change value for "Initialize Variables"** pop-up window. From this window, you define variables and provide a value for each of them. These variables are accessible throughout the process.

Initialize Variables tasks have the following properties:

Property	Description
ID	Unique id for this element.
Name	A name for this element.
Documentation	A description of this element.
Initialize Variables	Link to open the popup window which allows to add/create/delete/initialize process variables.
Enable Variable Type	Enables variable type conversions.

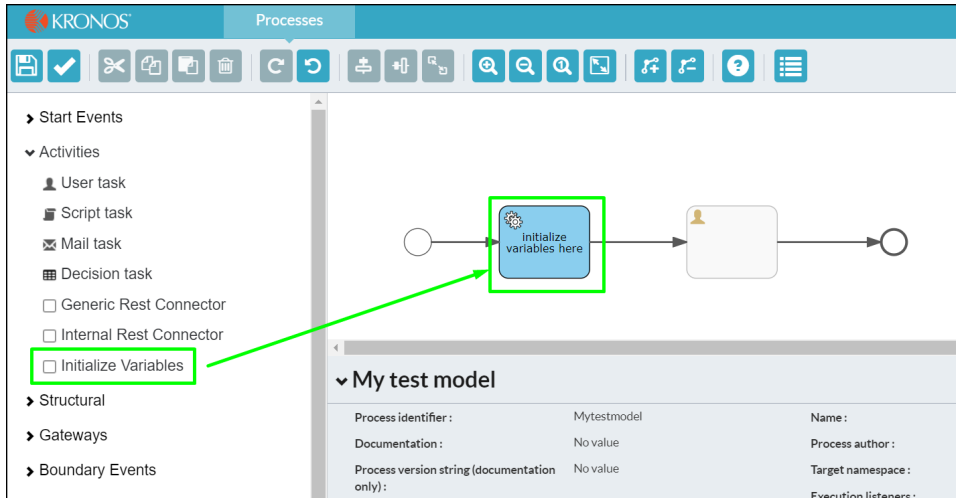
Note:

The Initialize Variables task:

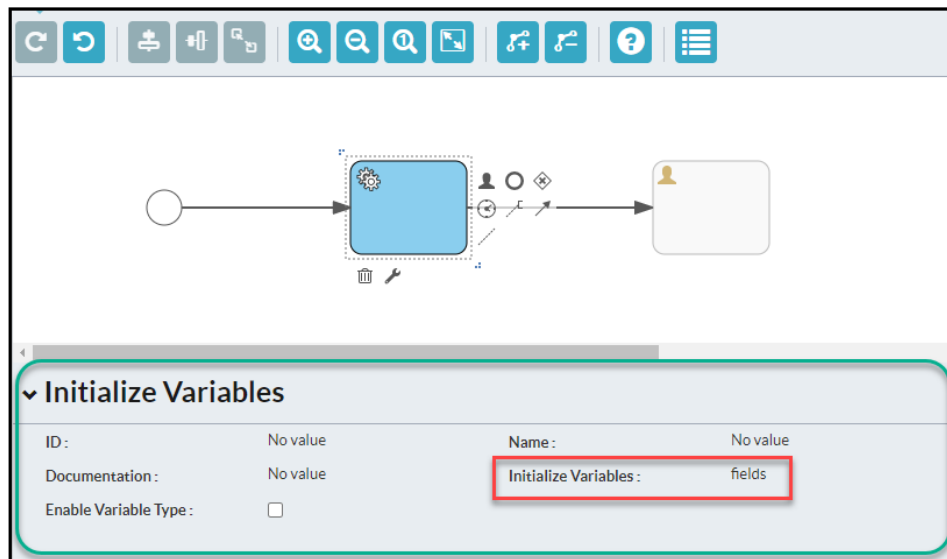
- Does not override the values of the process variables previously initialized anywhere in the flow before its own execution.
- Does not override the values of variables that are passed as parameters of the "Initiate Business Process" Rest API.
- Can be located anywhere in the process flow between Start task and End task.
- Can be used multiple times in the process flow.

Use the Initialize Variables task

To use the Initialize Variables task, open Workflow Designer, expand the **Activities** group, select **Initialize Variables**, drag it to the workspace, and attach it to a process.



1. Select **Initialize Variables** from the workspace, then click **Initialize Variables** in the Properties panel.



2. Select **Enable Variable Type** if you want the Initialize Variables task to convert variables to their respective data types.

Note: If conversion to the specified data type is not possible, the variable is converted to String. If the date cannot be converted to the specified format, or if the format is invalid, the date is converted to String.

3. In the **Change value for "Initialize Variables"** box, click + and enter the following:
- **Variable name** – Enter the name of the variable.
 - **Variable type** – String, integer, boolean, people, group, or date.
 - **Value** – Enter the value.
 - **Date Format** – If you have specified date for the variable type, optionally specify the date format you want to use. If you do not specify a format, the system uses the default: d-M-yyyy.

Note: The **Date Format** field is displayed only if you choose date for the **Variable type**.

Variable name	Variable type	Value
	date	

Variable name *

Enter variable name

Variable type

date

Value *

Enter variable value

Date Format

Enter date format (default format: d-M-yyyy)

Cancel Save

To create additional variables, click + again and provide the name, type, and value of the next variable. For example, you could create the following variables:

Variable Name	Variable Type	Value
question1	string	Enter your email
question2	string	Enter your phone number
selected_question	string	
error_message	string	

- When finished, click **Save**.
- The **Initialize Variable** field in the property panel displays the number of variables (fields) defined.

Note:

- The **Initialize Variables** activity is used to initialize variables when the Business Process is initialized regardless of where or how many times the activity appears in the flow.
- The **Initialize Variables** activity does not overwrite the value of any variables that are initialized as part of the call to the Business Process (for example, when an API is used to launch the Business Process).

Default variables

A number of components within ADP Workforce Manager use Business Processes to complete specific tasks. Each of these components include a predefined set of initialization variables. You access these variables from **Administration > Business Processes** in the Main Menu.

Platform and Control Center

Execution Variable Name	Description
initiator	
employee	
resourceId	
isCCRequest	
originator	
web_server_name	
web_server_protocol	

Execution Variable Name	Description
server_name	
application_context	
server_protocol	
server_language	
originator_kreference	
server_port	
wfc_url	
isReassignable	
ext_tenantId	
resourceIdType	
kronos_tid	
WFD_X_Forwarded_For	

Requests

Execution Variable Name	Description
initiator	Employee number initiating the request
employee	Employee number initiating the request
resourceId	
isCCRequest	
originator	Employee number initiating the request
StartDate	Start date of the request
PersonNumber	Employee number for the employee for whom the request is submitted
CurrentApproverID	Employee ID initiating the request
RequestItem	The request object
QueryDateSpan	The date span object of the request
web_server_name	Web server name
web_server_protocol	Web server protocol
server_name	Workflow engine server name
application_context	Workflow engine application context attribute
server_protocol	Workflow engine server protocol
server_language	Workflow engine server language
originator_kreference	Workflow attribute indicates originator's KPreference when the user initiates the workflow
server_port	Workflow engine server port
wfc_url	Workflow engine wfc_url attribute
isReassignable	Workflow engine is Reassignable attribute
ext_tenantId	Workflow engine ext_tenantID attribute

Attestation

Execution Variable Name	Description
initiator	Employee ID initiating the attestation
attestationProcessId	ID of attestation process that's initiating the workflow execution
timeFrame	Timeframe associated to the button that started the Attestation process. Defaults to 0 (previous pay period).
minutesSinceLastPunch	Minutes elapsed since last punch
punchTime	Last punch time (employee timezone)
estimatedPunchTime	Estimated punch time (UTC)
empDstStart	Employee time zone DST start
empDstEnd	Employee time zone DST end
userAgentHeader	HTTP user agent header
transferString	Transfer string selected by the user (if any) before starting attestation by clicking on an Attestation button
xForwardedForHeader	Standard HTTP header for xForwardedFor
empTimezoneOffset	Employee timezone offset
empTimezoneId	Employee timezone ID
estimatedPunchTimeInEmployeeTimezone	Estimated punch time in employee's time zone
todayDate	Today's date for employee
submitForManagerApproval	True if employee requires manager approval for timecard edits
enableFixMissingPunchWorkflow	Is fix missing punch enabled
predictiveSchedulingRequest	Is predictive scheduling request
enablePredictiveSchedulingWorkflow	Is predictive scheduling flow is enabled
shiftStartOrTodayDate	Shift start date for in-progress shift. Return current date if no shift is in progress.
positionId	PositionID for the current punch
userEnteredPosition	User- entered position true/false
positionName	Position name for the current punch

Execution Variable Name	Description
managerPersonId	Manager person for current position (or person - if single position employee)
manualTimeEntryAttestationConfigured	Is employee configured for manual time entry attestation

UDM

Execution Variable Name	Description
initiator	UDMUSER1/UDMUSER2
Offline	Is offline request from device or not
badgeNumber	Badge Number of the employee
deviceIdVar	Device ID
udmroute	For UDM routing
attestationButtonId	Attestation button ID
employeeId	EmployeeId for API calls
selectedId	Device selection
attestationOfflineFlowId	Deprecated
clockTimeVar	CLOCK Date time UTC
isSingleMessageVar	Used for 4500 single message req
is4500	Is device type 4500
authHeaderOnBehalfVar	Deprecated
isSummaryRequired	Used in some models for DX feature for summary
isMultiRequest	Used for DX feature for perform multiple submits
includeCalendar	Used for DX
deviceLocale	Device locale
optionalSubmit	
backSubmit	Back button press indicator
laborTransferVar	Attestation labor transfer value from button
jobsVar	

Execution Variable Name	Description
authHeader	Deprecated
attestationTimecardPeriod	Timeframe from button
disableNotifications	To mark email notification on user tasks as false
promptForPosition	Non-Attestation always true, Attestation value comes from button.
timeOffTimeOutMS	For RTO timeout value for submit API
timezoneIdVar	Device time zone ID
timezoneNameVar	Device time zone Name
timezoneOffsetVar	Device time zone offset value
timezoneDstStartVar	DST start time
timezoneDstEndVar	DST end time
timezoneTypeVar	timezone type
deviceAutoAdjustDstVar	Is DST
timezoneAdjustedClockTime	Time zone adjusted clock time.

Attestation Text Substitution

The Attestation Text Substitution task has been implemented to avoid any circular dependency between a workflow and other services such as WFM. The service task searches for the patterns in the task form JSON provided by the user. If found, the text is substituted and the form JSON is updated with the substituted text. If the pattern is not found, the form JSON is not updated and the output remains the same.

Attestation Text Substitution tasks have the following properties:

Property	Description
Name	A name for this element. Used to substitute a text pattern with an execution variable value in the form labels. It is a custom implementation.
Documentation	A description of this element.
ID	Unique id for this element.
Task Form JSON	The variable name that contains the JSON form structure.
Output Variable	The variable name that contains the substituted text.

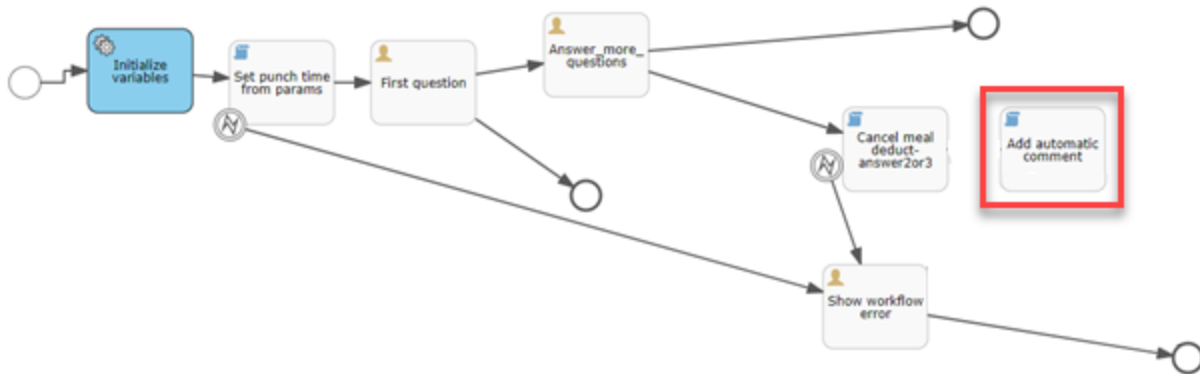
Configure automatic comments for manual time entry Attestation workflow models

Use this procedure to configure manual time entry Attestation models to add automatic comments to a timecard punch. You can use this procedure for any Attestation workflow model that supports manual time entry; these models have “for Manual Time Entry” appended to their names.

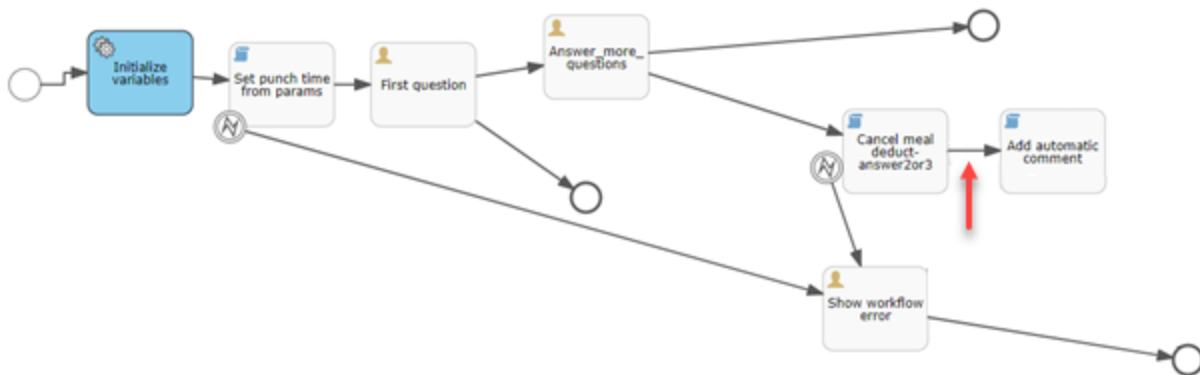
Note: Before you begin, any comment you specify using this procedure must be defined as a punch category comment on the Comments page (**Main Menu > Administration > Application Setup > Common Setup > Comments**) before you deploy the model. See the Comments topic in the ADP Workforce Manager online help for more information.

1. In ADP Workforce Manager, go to **Main Menu > Administration > Application Setup > Attestation > Attestation Models**.
2. On the Attestation Models page, copy the manual time entry model to which you want to add automatic commenting.
3. Select the copied model and click **Edit**.
The model is displayed in the Workflow Designer.
4. In Workflow Designer, click **Visual Editor**.
The model is displayed in the Visual Editor.
5. In the model, locate and select the Initialize Variables task.
6. In the properties panel, select the **Initialize Variables** field.
The Change value for "Initialize Variables" window appears.
7. Add the automatic comment you want the system to display by specifying values in these fields:
 - **Variable name:** In this example, and in the script that follows in step 13, the variable name is "comment1".
 - **Variable type:** Set to string.
 - **Value:** Enter the comment. The value must be the same as a comment defined on the Comments page in Common Setup.
8. Save your changes.
9. In the Visual Editor, select **Activities > Script task**, and add a Script task to the model in the desired location.

Note: Typically, you want to insert the Script task on the path after the form on which you wish to display the comment based on the employee's answer. In this illustration, the Script task is named "Add automatic comment", and it is added after the "Cancel meal deduct answer2or3" task.



10. Add a line connecting the preceding task to the Script task you just added:



11. In the model, locate and select the Script task you just added.
12. In the properties panel, click the **Script:** field.
The Change value for "Script" window appears.
13. Enter this script in the Change value for "Script" window:

```

import groovy.json.JsonOutput
import groovy.json.JsonSlurper
import org.activiti.engine.delegate.BpmnError
try {
    setComment(execution.getVariable('comment1') as String)
} catch (err) {
    throwError(err.message)
}
def setComment(String commentName) {
    if (commentName != null && commentName != '') {
        def punchJson = execution.getVariable('manualTimeEntryPunch')
        def punchObj = new JsonSlurper().parseText(punchJson as String)
        def newComment = ['comment': ["name": commentName]];
        def commentsNotes = punchObj.getAt('commentsNotes')
        boolean commentAdded = false;
        if (commentsNotes) {
            boolean commentNameAlreadyExists = false;
            for (c in commentsNotes) {
                if (commentName.equals(c.getAt('comment').getAt('name'))) {
                    commentNameAlreadyExists = true;
                    break;
                }
            }
            if (!commentNameAlreadyExists) {
                commentsNotes += newComment;
                punchObj['commentsNotes'] = commentsNotes
                commentAdded = true
            }
        } else {
            punchObj['hasComments'] = true;
            punchObj['commentsNotes'] = [newComment];
            commentAdded = true
        }
        if (commentAdded) {
            execution.setVariable('manualTimeEntryPunchStatus', "PUNCH_
UPDATED")
            def serializedPunch = JsonOutput.toJson(punchObj)
            execution.setVariable('manualTimeEntryPunch', serializedPunch)
        } else {
    
```

```

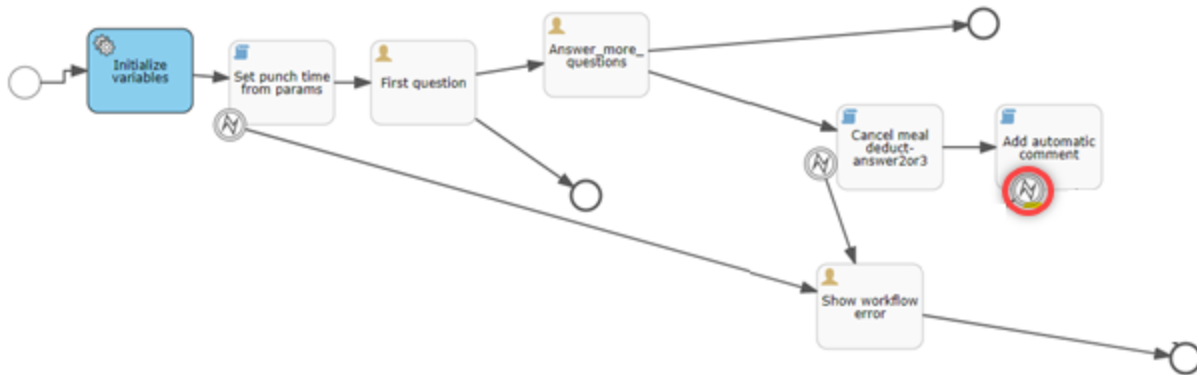
        execution.setVariable('manualTimeEntryPunchStatus', "PUNCH_
NO_CHANGES")
    }
    } else {
        execution.setVariable('manualTimeEntryPunchStatus', "PUNCH_NO_
CHANGES")
    }
}
def throwError(String errorMsg) {
    execution.setVariable('manualTimeEntryPunchStatus', 'PUNCH_CANCELLED')
    execution.setVariable("error_message", errorMsg)
    throw new org.activiti.engine.delegate.BpmnError("error_message",
errorMsg);
}
}

```

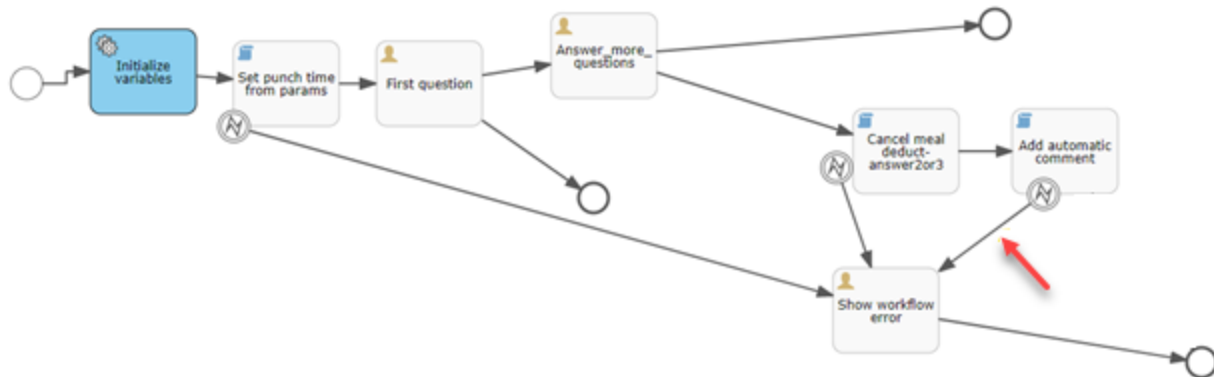
- Find this line in the script, and substitute the value 'comment1' with the variable name you created in step 7:

```
setComment(execution.getVariable('comment1') as String
```

- Click **Save**.
- In the Visual Editor, select **Boundary Events > Boundary Error Event**, and add a boundary error event to the Script task you added previously:



- Add the Sequence flow from the Boundary error event to the Error task:



18. Validate the model and address any issues.
19. Save the model.
20. In ADP Workforce Manager, deploy and test the model.

Chapter 4: Structural

The Structural group includes a number of subprocesses, which are used for repeated or reusable components or for compensation tasks. When collapsed, subprocesses hide complexity and makes the overall Business Process diagram look tidy. You can jump to a subprocess by using a [boundary event](#) to interrupt a process.

Subprocesses in the Structural group include:

- [Subprocess on page 73](#)
- [Collapsed Subprocess on page 75](#)
- [Embedded subprocess on page 76](#)
- [Event subprocess on page 78](#)
- [Call activity subprocess on page 78](#)
- [Execute Business Process By ID on page 81](#)
- [Submit User Task By ID on page 82](#)

Subprocess

A subprocess is a single activity that contains activities, gateways, and events that form a process. A subprocess is completely embedded inside a parent process.

You can use a subprocess to create a new scope for events. Events that are "thrown" during execution of the subprocess can be "caught" by [Boundary events](#) on the boundary of the subprocess, creating a scope for that event limited to just the subprocess.

Subprocesses must have the following characteristics:

- A subprocess has exactly one None start event. No other start event types are permitted. A subprocess must have at least one End event.
- Sequence flow cannot cross subprocess boundaries.

- When collapsed, a subprocess hides complexity and makes the overall Business Process diagram look tidy.
- Use subprocesses for repeated or reusable components, or for compensation tasks.
- Select the plus sign to expand and show the subprocess.
- Boundary events can interrupt a process and jump to a subprocess.

Subprocesses have the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Asynchronous	(Advanced) Define this task as asynchronous. This means the task will not be executed as part of the current action of the user, but later. This can be useful if it's not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Multi-Instance type	Determines if this task is performed multiple times and how. The possible values are: <ul style="list-style-type: none"> • 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)	The number of times the task is to be performed.
Collection (Multi-instance)	The name of a process variable that is a collection. For each item in the collection, an instance of this task will be created.
Element variable (Multi-instance)	A process variable name that 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. You can specify an expression here to be evaluated each time an instance ends. If the expression evaluates to true, all remaining instances are destroyed and the multi-instance activity ends.

Collapsed Subprocess

You use a collapsed subprocess to add an existing process from your available process definitions as a subprocess to the process definition you are currently editing.

When you drag a collapsed subprocess from the Stencil Set to your canvas, and click on the **Referenced Subprocess** property, a visual list of the process definitions you can access to displays. When you select a process to add to the current process definition, the process selected must have the following attributes:

- One None Start event
- No other Start Event types
- At least one End Event.

Note: During process instance execution, there is no difference between a collapsed or embedded subprocess. They both share the full process instance context (unlike the [call activity](#)).

Collapsed subprocesses have the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element instance.
Referenced Subprocess	The process definition this collapsed subprocess contains.
Asynchronous	(Advanced) Define this task as asynchronous. This means the task is not executed as part of the current action of the user, but later. This can be useful if it's not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Multi-Instance type	Determines if this task is performed multiple times and how. The possible values are: <ul style="list-style-type: none">• 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	The number of times the task is to be performed.

Property	Description
(Multi-instance)	
Collection (Multi-instance)	The name of a process variable that is a collection. For each item in the collection, an instance of this task will be created.
Element variable (Multi-instance)	A process variable name that 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. You can specify an expression here to be evaluated each time an instance ends. If the expression evaluates to true, all remaining instances are destroyed and the multi-instance activity ends.

Embedded subprocess

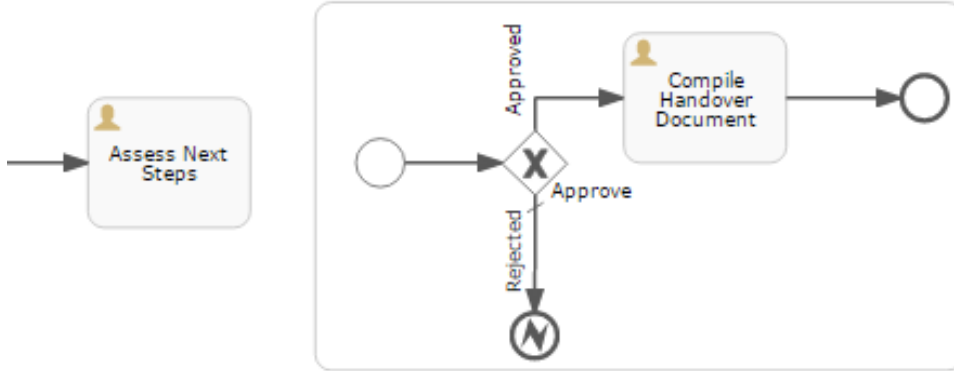
The purpose of an embedded subprocess is to clarify the diagram and boundary event behavior within the main process. It shares the variable scope with the container process.

To configure embedded subprocesses:

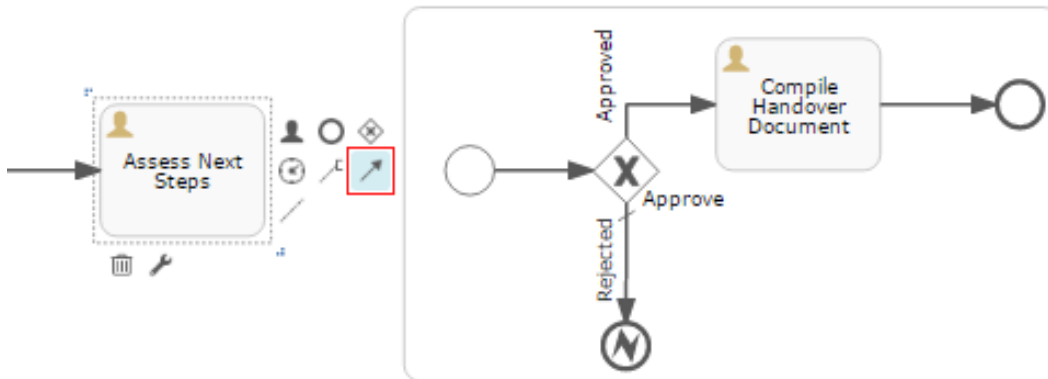
1. From the Stencil Set, select **Structural**, then click and drag a **subprocess** to the relevant position in the process.



2. Click and drag the lower-right corner of the subprocess to make the box larger.
3. From the Stencil Set, select **Start Events**, then click and drag any type of **Start Event** to the inside of the subprocess.
4. Build the subprocess with [gateways](#), [events](#), and [forms](#) as required, for example:



5. Select the event just before the subprocess. Click and drag the sequence flow to the subprocess.



6. Click **Validate the Model** . Correct any errors.

Note: Models that fail validation **cannot** be deployed.

7. Click **Save the Model** .

- Verify the settings.
- Click Save and Close Editor or Save.

8. Deploy, run, and test the model.

Subprocess types compared

	Embedded	Call Activity	Event
Reusable	No	Yes	No

	Embedded	Call Activity	Event
Independent version cycle	No	Yes	No
Can start a main process	No	Yes	No
Can share context with a main process	Yes	No	Yes
Can be deployed with the calling process	Yes	Maybe	Yes
Maintenance	Low	High	Low

Event subprocess


An event subprocess is triggered by an event.

1. From the Stencil Set, select **Structural**, then click and drag an **Event subprocess** to the relevant position in the process.
2. Click and drag the lower-right corner of the subprocess to make the box larger.
3. In **Properties**, enter a **Name**.
4. From the Stencil Set, select **Start Events** and click and drag an event-triggered start event to the inside of the subprocess:

 **Note:** None start events are not supported.

 Start timer event

 Start signal event

 Start error event

 Message start event

5. Build the subprocess with [gateways](#), [events](#), and [forms](#) as required.
6. Deploy, run, and test the model.

Call activity subprocess

A **call activity or external subprocess** runs external code or another Business Process. The main difference between a subprocess and a call activity is that the call activity does not share context with the process

instance. Process variables are explicitly mapped between the process instance and the call activity. Characteristics of all activity subprocesses are:



- Standalone process with independent instance and execution.
- Used to set aside processing, variables, and output only the results.
- Passes parameters inward and outward.
- Can be a main process that can run alone or can be reused by multiple processes.


A Call activity subprocess has the following properties:


Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Called element	This is the identifier of the process definition that should be called.
In parameters	Configures the process variables that are mapped into the called process instance when it is executed. It is possible to copy values directly (using the source attribute) or with an expression (using the source expression attribute) in a target variable of the called process instance.
Out parameters	Configures the process variables that are mapped from the called process instance into the parent process instance.
Asynchronous	(Advanced) Define this task as asynchronous. This means the task is not executed as part of the current action of the user, but later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Execution listeners	Execution listeners configured for this instance. An execution listener is a piece of logic that is not shown in the diagram and can be used for technical purposes.
Multi-Instance type	Determines if this task is performed multiple times and how. 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)	The number of times the task is to be performed.

Property	Description
Collection (Multi-instance)	The name of a process variable that is a collection. For each item in the collection, an instance of this task is created.
Element variable (Multi-instance)	A process variable name 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. You can specify an expression here to be evaluated each time an instance ends. If the expression evaluates to true, all remaining instances are destroyed and the multi-instance activity ends.


Configure a Call Activity subprocess

1. From the Stencil Set, select **Structural**, then click and drag a **Call Activity** element to the relevant position in the process.
2. Select the call activity.
3. In **Properties**:
 - a. Enter a **Name**, for example: `Call Evaluate Other Options Process`
 - b. In **Called Element**, enter the ID of the relevant subprocess. for example, `EvaluateOtherOptions` or `send_alert_notification`
4. Configure *input parameters* from the parameters that pass into the external subprocess .
 - a. Select **In Parameters**.
 - b. Click Add .
 - c. In **Source**, either pass input parameters directly or derive them by an expression. The variable is passed to the called subprocess from the top-level process, for example, `initiator`
 - d. Repeat for other variables, for example, `estimatedcosts` and `nextsteps`
 - e. Click **Save**.
5. Configure *output parameters* from the parameters that pass from the external subprocess to the main process.
 - a. Select **In Parameters**.
 - b. Click Add .
 - c. In **Source**, either explicitly align output parameters with subprocess variables or define them by an expression. Output variables override parent variables.

- d. Repeat for other variables.
- e. Click **Save**.
- 6. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

- 7. Click **Save the Model**
- 8. Deploy, run, and test the model.

 **Caution:** Check for timing delays during processing.

Execute Business Process By ID

The Execute Business Process By ID service task is used to execute a Business Process and return the form corresponding to its first user task in the flow. It is a custom implementation that has been implemented to avoid any circular dependency between a workflow and other services.

The service task executes the Business Process on the basis of process deploymentID and the payload (process initialization variables) provided. The response can be retrieved in a unique response variable and used thereafter by the Submit User Task service task.

The Execute Business Process By ID service task has the following properties:

Property	Description
Name	A name for this element.
Documentation	A description of this element.
ID	Unique ID for this element
Process Deployment ID	The deployment ID of the Business Process to be executed.
Process Initialization variables JSON	The JSON payload to execute the Business Process. Format: <pre>{ "name": "initializationVariableName1", "value": "initializationVariableValue1" }, { "name": "initializationVariableName2", "value": "initializationVariableValue2" }</pre>
Response Variable	The variable to store the JSON response. The response is a JSON having the processInstanceID, an array of formStructureData with taskID, and an array of formValueData which consists instance variable of child's Business Process.

Submit User Task By ID

The Submit User Task By ID service task is used to execute a Business Process and return the form corresponding to its first user task in the flow. It is a custom implementation that has been implemented to avoid any circular dependency between a workflow and other services.

The service task executes the Business Process on the basis of process deploymentID and the payload (process initialization variables) provided. The response can be retrieved in a unique response variable and used thereafter by the Submit User Task service task.

The Submit User Task By ID service task has the following properties:

Property	Description
Name	A name for this element.
Documentation	A description of this element.
ID	Unique ID for this element
Process Deployment ID	The deployment ID of the Business Process to be executed.
Process Initialization variables JSON	The JSON payload to execute the Business Process. Format: <pre>{ "name": "initializationVariableName1", "value": "initializationVariableValue1" }, { "name": "initializationVariableName2", "value": "initializationVariableValue2" }</pre>
Response Variable	The variable to store the JSON response. The response is a JSON having the processInstanceID, an array of formStructureData with taskID, and an array of formValueData which consists instance variable of child's Business Process.

Chapter 5: Gateways

Gateways are responsible for controlling how a Business Process flows. In a process, the work to do and the output may vary under different external or internal conditions. For example, a discount will only be offered to a VIP buyer but not to anyone else. A gateway is where conditions are evaluated and the decision is made.

The following types of gateway are available from the Stencil Set:

- [Exclusive gateway on page 83](#)
- [Parallel gateway on page 88](#)
- [Inclusive gateway on page 89](#)
- [Event gateway on page 90](#)

Exclusive gateway

An Exclusive gateway (also called the XOR gateway) is used to model a decision in the process. When the execution arrives at this gateway, all outgoing sequence flow are evaluated in the order in which they are defined. The sequence flow that the condition evaluates to true is selected for continuing the process.


Essentially, an exclusive gateway evaluates the current state of the process and decide which mutually exclusive path to follow. For example, If or Else, Yes or No, True or False, Enable or Disable.

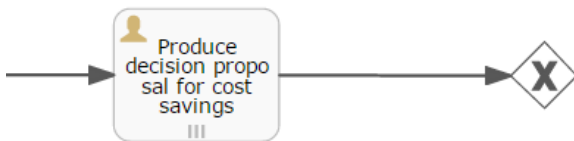
An Exclusive gateway has the following properties:

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Asynchronous	(Advanced) Define this task as asynchronous. This means the task is not executed as part of the current action

Property	Description
	of the user, but later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none are executed at the same time. This is useful to solve race conditions.
Flow order	Select the order in which the sequence flow conditions are evaluated. The first sequence flow that has a condition that evaluates to true (or has no condition) will be selected to continue.

To configure an Exclusive Gateway:

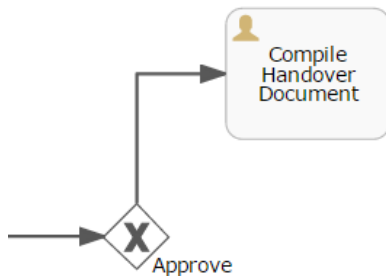
1. Select the task that is just before the point where you want to put the gateway.
2. Select and connect an **Exclusive Gateway** .



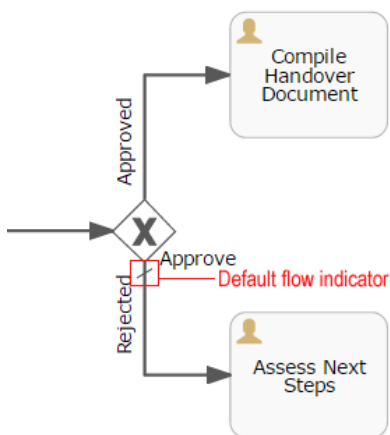
3. Follow the gateway with two paths.
 - a. Select the gateway.
 - b. In **Properties**, enter a **Name**, for example: *Approve?*.
 - c. Do one of the following:
 - Select the task, end, or intermediate capture event, or another gateway that is the next step on the path.



- Select and drag an event from the list of events to the workspace. Connect a sequence flow from the gateway to the event.
- d. Select the event.
- In **Properties**, enter a **Name**, for example: `Compile Handover Document`.
 - In **Assignments**, select **Assigned to process initiator**.
 - Select and drag the event to the top of the diagram to make room for another event from the gateway.
 - Select and drag the sequence flow so that the diagram looks tidy.



- e. Select the sequence flow. In **Properties**, enter a **Name**, for example: `Approve`
- f. Repeat for the other path.
4. Set one output flow as the default path to follow for a false result.
- a. Select that sequence flow.
- b. Select **Default Flow**. The default flow is marked with a short line at an angle.



5. Define the flow conditions from the exclusive gateway.
 - a. Select the sequence flow from the gateway to the first event.
 - b. In **Properties**, select **Flow Condition**.
 - c. In **Flow Condition**, select **Simple > Form Field**.
 - d. Select a field and operator, and enter a value.

Example: Select **Approval Decision - id** and **equals**. Enter **approved**.

Alternatively, you can define the flow condition in an expression:

- `${variable == 'variableValue'}`
- `variable` is any execution variable, including variables associated with the value of a field in a form
- `variableValue` is the value of the variable that activates this branch of the logic.


 **Example:**

- Default flow condition: `${Approve == 'false'}`
- Exception flow condition: `${Approve == 'true'}`



- e. Click **Save**.
 - f. Repeat for the other event.
6. Create forms for the user tasks.
 - Select a **User Task** .
 - a. In the **Property** panel at the bottom, select **Referenced Form**.
 - b. Select **New Form**.
 - c. Enter the **Form Name**.
 - d. From **Stencil**, select **ADPFormStencil**.
 - e. Click **Create Form**.
 - Click the **Forms** tab.
 - Select the form.
 - Click **Form Editor > Design** tab. See .
 - Repeat for the other user task.

Example


User task 1: Approved sequence flow



- **Name:** Compile Handover Document
-  **Display text – Name =** Approved Message; **Text to display =** Approved. Complete and attach the Handover Document.
- **Attach – Label =** Handover Document. **Select Required.**

User task 2: Rejected sequence flow


- **Name:** Assess Next Steps
-  **Display text: Name =** Rejected Message; **Text to display =** Rejected. Enter next steps.
-  **Multiline text: Label =** Next Steps. **Select Required.**


7. a. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

- b. Click **Save the Model** .
 - Verify the settings.
 - Click **Save and Close Editor** or **Save**.
8. Attach the form to the user task.
 - a. Select the **Processes** tab.
 - b. Select the process.
 - c. Select **Visual Editor** .
 - d. Select the user task.
 - e. In **Properties**, select **Referenced Form**.
 - f. Select the form.
 - g. Click **Save**.
 - h. Repeat for the other user task.

9. a. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

- b. Click **Save the Model** .
 - Verify the settings.
 - Click **Save and Close Editor** or **Save**.
10. Deploy, run, and test the model.


Parallel gateway

The Parallel gateway can be used to introduce concurrency in a process model, which allows the process to fork into multiple paths of execution or join multiple incoming paths of execution.

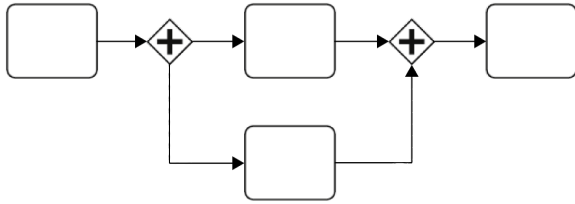
The functionality of the parallel gateway is based on the incoming and outgoing sequence flow:

- **fork** – All outgoing sequence flows are followed in parallel, creating one concurrent execution for each sequence flow.
- **join** – All concurrent executions arriving at the parallel gateway wait in the gateway until an execution has arrived for each of the incoming sequence flow. Then the process continues past the joining gateway.

Note that a parallel gateway can have both fork and join behavior, if there are multiple incoming and outgoing sequence flows for the same parallel gateway. In that case, the gateway will first join all incoming sequence flow, before splitting into multiple concurrent paths of executions.

 **Note:** An important difference with other gateway types is that the parallel gateway does not evaluate conditions. If conditions are defined on the sequence flow connected with the parallel gateway, they are simply neglected.

A parallel gateway branches the process into concurrent paths; the process follows all paths. It requires two parallel gateway objects, one to split and the other to merge. It does not evaluate conditions or events.



Tip: Use a parallel gateway **only** if the logic **cannot** go wrong. If a path stops, the parallel gateway will wait forever, and the process will stop.

Property	Description
ID	A unique identifier for this element.
Name	A name for this element.
Documentation	A description of this element.
Asynchronous	(Advanced) Define this task as asynchronous. This means the task is not executed as part of the current action of the user, but later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Flow order	Select the order in which the sequence flow conditions are evaluated. Note that for a parallel gateway this is not important, as conditions are not evaluated.

Inclusive gateway


The Inclusive Gateway can be seen as a combination of an exclusive gateway and a parallel gateway. Like an exclusive gateway, you can define conditions on outgoing sequence flows and the inclusive gateway will evaluate them. But the main difference is that the inclusive gateway can take more than one sequence flow, like the parallel gateway.

The functionality of the inclusive gateway is based on the incoming and outgoing sequence flow:

- **fork**— All outgoing sequence flow conditions are evaluated and for the sequence flow conditions that evaluate to true the flows are followed in parallel, creating one concurrent execution for each sequence flow.
- **join**— All concurrent executions arriving at the inclusive gateway wait in the gateway until an execution has arrived for each of the incoming sequence flows that have a process token. This is an important difference with the parallel gateway. So in other words, the inclusive gateway will only wait for the incoming sequence flows that will be executed. After the join, the process continues past the joining inclusive gateway.

Note that an inclusive gateway can have both fork and join behavior, if there are multiple incoming and outgoing sequence flow for the same inclusive gateway. In that case, the gateway will first join all incoming sequence flows that have a process token, before splitting into multiple concurrent paths of executions for the outgoing sequence flows that have a condition that evaluates to true.

An inclusive gateway breaks the flow into one or more flows. The resulting flows are not mutually exclusive but can be concurrent and at least one condition must be true.

 **Example:** Act in one or more ways based on survey results.

Property	Description
ID	A unique identifier for this element instance.
Name	A name for this element instance.
Documentation	A description of this element instance.
Asynchronous	(Advanced) Define this task as asynchronous. That is, the task is not executed as part of the current action of the user, but later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. That is, when there are multiple asynchronous elements of the same process instance, none are executed at the same time. This is useful to solve race conditions.
Flow order	Select the order in which the sequence flow conditions are evaluated. This is of less importance as for the exclusive gateway, as all outgoing sequence flow conditions will be evaluated anyway.

Event gateway

An event gateway makes a decision based on events. Each outgoing sequence flow of the gateway must be connected to an intermediate catching event. When process execution reaches an event gateway, the gateway acts like a wait state: execution is suspended. In addition, for each outgoing sequence flow, an event subscription is created.

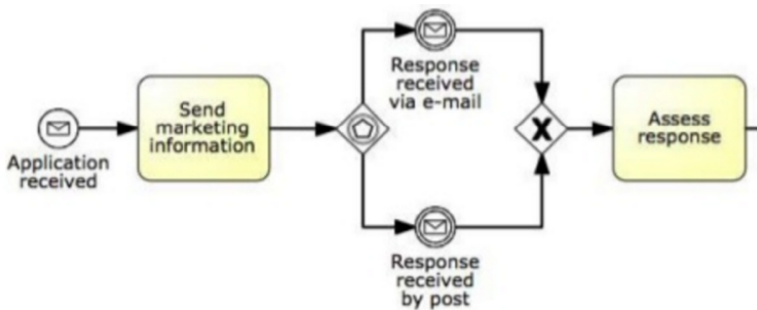
Note that the sequence flows running out of an event gateway are different from ordinary sequence flows. These sequence flows are never actually "executed." Instead, they allow the process engine to determine which events an execution arriving at an event gateway needs to subscribe to. The following restrictions apply:

- An event gateway must have two or more outgoing sequence flows.
- An event gateway must only be connected to elements of type [Intermediate Catch Event](#) only.
- An Intermediate Catch Event connected to an event gateway must have a single incoming sequence flow.

Like an exclusive gateway, an event gateway evaluates the state of a process and breaks the flow into mutually exclusive paths. However, an event-based gateway evaluates which event occurred, not which condition was met. The logic is not based on data but on the following events.

- Only one output sequence is followed which is the first event to occur (XOR).
- An event gateway is a wait location and connects **only** to intermediate timer or message catching events.

Example:



Event gateways have the following properties:

Property	Description
ID	A unique identifier for this element instance.
Name	A name for this element instance.
Documentation	A description of this element instance.
Asynchronous	(Advanced) Define this task as asynchronous. This means the task is not executed as part of the current action of the user, but later. This can be useful if it is not important to have the task immediately ready.
Exclusive	(Advanced) Define this task as exclusive. This means that, when there are multiple asynchronous elements of the same process instance, none will be executed at the same time. This is useful to solve race conditions.
Flow order	Select the order in which the sequence flow conditions are evaluated.

Chapter 6: Events

Events are used to model something that happens during the workflow. There are two main event categories: catching or throwing events.


- **Catching** – When process execution arrives in the event, it will wait for a trigger to happen or to "catch."
- **Throwing** – When process execution arrives in the event, a trigger is fired or "thrown."


An event subprocess is triggered by an event. You can use an event subprocess in a main process or in any subprocess. See [Example: Add an escalation timer and signal event to a workflow on page 95](#) and [Event subprocess on page 78](#) for examples.


- [Boundary Events](#)
- [Intermediate Catching Events](#)
- [Intermediate Throwing Events](#)
- [End Events](#)


Boundary Events

Boundary events are assigned to other elements such as service tasks and user tasks. Boundary events are "catching" events that are attached to an activity. This means that while the activity is running, the event is listening for a certain type of trigger. When the event is caught, the activity is interrupted and the sequence flow going out of the event are followed. Interrupt a process or subprocess at an activity. The object is attached to the boundary of the activity


 **Boundary timer event** – When a task does not finish within the defined time, jump the process to a timer start or timer intermediate catching event. Use to move a process to an escalation path.

 **Boundary signal event** – Listens for signal intermediate throwing events, then jumps the process to a signal start or signal intermediate catching event.


 **Boundary error event** – Catches a process error (BPMNError), interrupts the task, jumps the process to an error start or error catching event.

 **Boundary compensation event** – An attached intermediate catching cancel on the boundary of a transaction subprocess, or boundary cancel event for short, is triggered when a transaction is cancelled.

When the cancel boundary event is triggered, it first interrupts all executions active in the current scope. Next, it starts compensation of all active compensation boundary events in the scope of the transaction. Compensation is performed synchronously – the boundary event waits before compensation is completed before leaving the transaction. When compensation is completed, the transaction subprocess is left using the sequence flow(s) running out of the cancel boundary event.


 **Note:**


- Only a single cancel boundary event is allowed for a transaction subprocess.
- If the transaction subprocess hosts nested subprocesses, compensation is only triggered for subprocesses that have completed successfully.
- If a cancel boundary event is placed on a transaction subprocess with multi instance characteristics, if one instance triggers a cancellation, the boundary event cancels all instances.

 **Boundary message event** – Catches a message from an external system. Can be configured to cancel an attached activity.

Intermediate Catching Events

Conceptually, intermediate catch events are close to the boundary events, with the exception that they do not define a scope (the activity) for when the event is active. An intermediate catch event is active as long as the trigger has not happened. A boundary event on the other hand can be destroyed if the activity completed.

 **Intermediate timer catching event** – Pause the process or subprocess for a defined period of time. It is used to wait before proceeding or to move a process to an escalation path.

 **Intermediate signal catching event** An intermediate catching signal event catches signals with the same signal name as the referenced signal definition.

Note: Contrary to other events such as an error event, a signal is not consumed if it is caught. If you have two active signal boundary events catching the same signal event, both boundary events are triggered, even if they are part of different process instances.

Message intermediate event – Catch a message from an external system. Normally, message events link to external systems rather than between or within processes. Message events are point-to-point, not broadcast.

Intermediate Throwing Events

An intermediate throwing event is used to explicitly throw an event of a certain type.

Signal intermediate throwing event – Broadcasts a signal across the process to be caught by signal boundary events or signal intermediate catching events, for example, send a signal to be picked up in a subprocess. The signals are limited to the same tenant and system database, but do not have to run on the same process server. Normally, signals do not carry a payload unless configured to use an API.

End Events




An end event signifies the end (of a path) of a process or subprocess. An end event is always throwing, which means that when process execution arrives in the end event, a result is thrown. The type of result is depicted by the inner black icon of the event.

- End event** – A none end event ends the current path of execution.
- Error end event** – The current path of execution is ended and an error is thrown.
- Cancel end event** – Trigger tasks that run before cancelling the process
- Terminate end event** – End the current process and all subprocesses

Example: Add an escalation timer and signal event to a workflow

To add an escalation timer and signal event to a workflow, complete the following steps.

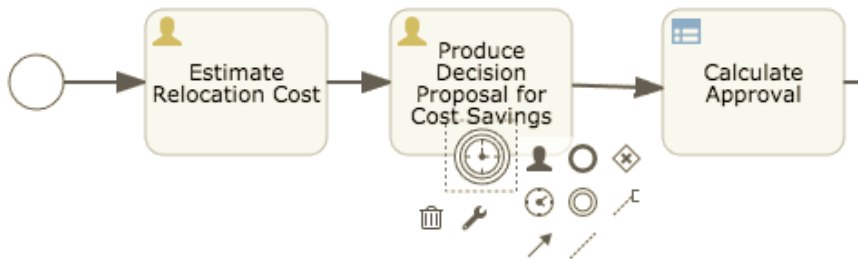
Step 1: Select a process to edit

1. Click **Main Menu**  > **Administration** > **Application Setup** > **Business Process Setup** > **Process Models**.
2. Select the model. Click **Edit** .
3. Click **Visual Editor** .

Step 2: Add an escalation timer

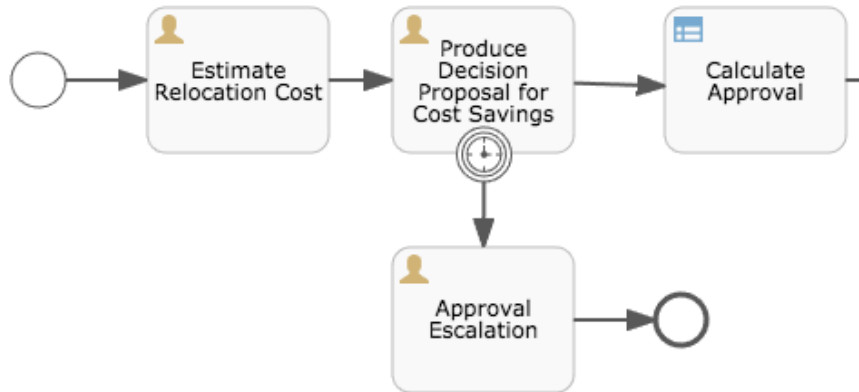
As an example, you could have a timer send notifications three times to a manager before it assigns an escalation task.

1. From the Stencil Set, open **Boundary Events** and click and drag a **Boundary Timer Event** to the relevant task in the process, for example:



2. Select the boundary timer event, then, in **Properties**:
 - a. Enter a **Name**.
 - b. Enter the **Time Cycle**. Use ISO 8601 format which is an international standard to exchange date and time-related data in order to avoid misinterpretation, especially between countries that use different conventions for numeric dates and times. The standard defines unambiguous representations of dates, times, and durations, for example:
For three repeats at 1-day intervals, enter `R3/PT24H`.
For testing, set the intervals to 30 seconds, that is, `R3/PT30S`.
 - c. Clear **Cancel Activity**.
3. From the Stencil Set, select **Activities**, then click and drag a **User Task** to the relevant position in the process.
 - a. Enter a **Name**, for example: `Approval Escalation`.
 - b. In **Assignment**, select **Assign to process initiator**, then click **Save**.
4. Connect a sequence flow from the timer boundary event to the user task.

5. Add an end event after the user task. For example, to add a timer boundary event with a user task and end event.



- a. Click **Validate the Model** . Correct any errors.

Caution: Models that fail validation **cannot** be deployed.

- b. Click **Save the Model** .
 - Verify the settings.
 - Click **Save and Close Editor** or **Save**.


6. Deploy, run, and test the model.

Example: Verify that at the `Produce Decision Proposal for Cost Savings` step, the `3 Approval Escalation` tasks repeat at 30-second intervals.

Step 3: Add a signal event

Add a signal throwing event immediately after the `Produce Decision Proposal for Cost Savings` task and a signal boundary catching event to the `Approval Escalation` task to cancel that task.

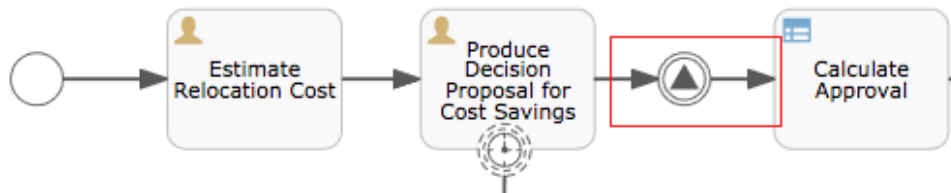
1. Select the Process Model. Click **Edit**.
2. Click **Visual Editor** .
3. Add a signal.
 - a. Click an open space.
 - b. In **Properties**, select **Signal Definitions**.

- c. Select Add Signal .
- d. Enter a unique **ID** and **Name**. Signal IDs must be unique within the system or cluster, for example:

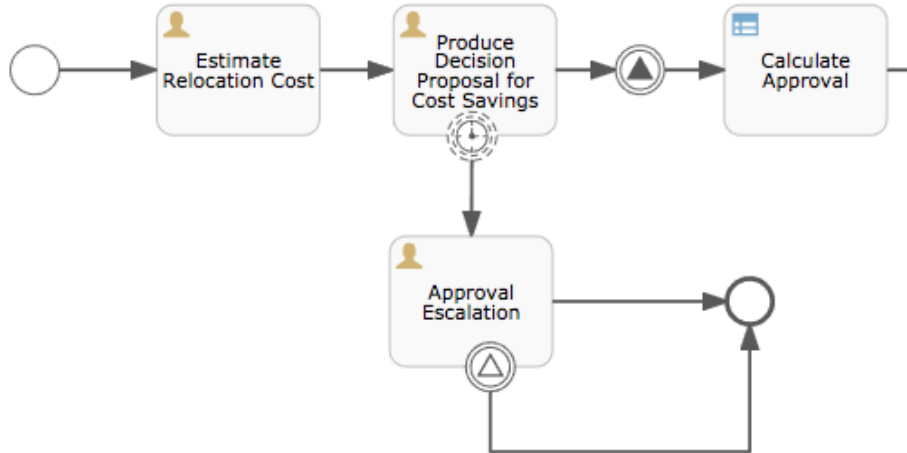

```
escalateApprovalCancel
```
- e. In **Scope**, select **Process Instance**.

Note: The default Scope for signal events is Global, which means that any process instance with listeners subscribed to the signal is triggered when that signal is thrown. Normally, this is **not** desired because it is preferable to catch **only** signals that are thrown in the current process instance.

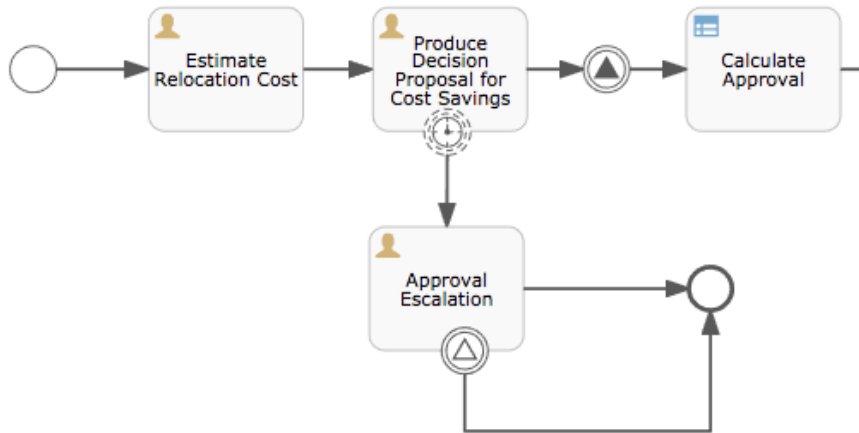
- f. Click **Save**.
4. From the Stencil Set, open **Intermediate Throwing Events** and click and drag an **Intermediate signal throwing event** between the relevant tasks in the process, for example:



5. Select the signal throwing event, then in **Properties**:
 - a. Enter a **Name**.
 - b. In **Signal Reference**, select the signal that you added so that any subscribed listeners catch this signal, for example: `escalateApprovalCancel`.
6. Cancel the tasks that are triggered by the boundary timer event.
 - a. From the palette, open **Boundary Events** and click and drag an **Boundary Signal Event** to the triggered tasks in the process. Example: The `Approval Escalation` task
 - b. Connect a sequence flow from the signal boundary event to the relevant end task




7. Select the signal boundary event, then in **Properties**:
 - a. Enter a **Name**.
 - b. Select **Cancel Activity**.
 - c. In **Signal Reference**, select the signal that the signal throwing event triggers, for example: `escalateApprovalCancel`.



8. a. Click **Validate the Model** . Correct any errors.

⚠ Caution: Models that fail validation **cannot** be deployed.

- b. Click **Save the Model** .
 - Verify the settings.
 - Click **Save and Close Editor** or **Save**.
9. Deploy, run, and test the model.

Listeners

Listeners implement hook points inside a process definition. These are triggered by events during workflow execution. There are several types of listeners:

- **Event listeners** – For specific event types in the process engine or process definition, define common behavior among multiple instances.

Execute custom script code or evaluate an expression when the following events occur:

- Start and end of a process instance
- A transition
- Start and end of an activity
- Start and end of a gateway
- Start and end of intermediate events
- End of a start event or start of an end event

Examples:

- Listener implementation = `log.info("*****Embedded subprocess has started");`
 - Type = expression
 - Event = start
- **Execution listeners** – Callback interface to be notified of execution events such as starting a process instance, ending an activity instance, or taking a transition.
 - **Task listeners** – Task listeners are similar to execution listeners, but is used to trigger a custom implementation upon the occurrence of a particular task.

Event listeners

Global event listeners that are triggered by task or execution events include the following:

- Defined for each process application.
- Used for functions that are required for all tasks or executions, for example, send email to task assignees for all user tasks.
- Event listeners must implement the `ActivitiEventListener` interface.

Events include the following:

- Entity changes
- Task and instance changes

- Membership changes
- Variable changes
- Incoming messages or signals
- Uncaught BPMN errors

Execution listeners

In a process definition, execution listeners define behavior for specific events **only** for that step or transition.

Script task execution listeners

Task and execution listeners execute Java classes that implement the `JavaDelegate` interface. Alternatively, a script-task execution listener is a class called:

`org.activiti.engine.impl.bpmn.listener.ScriptTaskListener`. Use listeners to do the following:

- Initiate and configure variables.
- Send notifications.
- Send messages.

Task listeners

Task listeners are put in the process definition for a user task. Similar to an execution listener, task listeners execute custom Java code or an evaluation expression at user-task events. Events include the following:

- Create
- Assign
- Complete

Examples

Example 1:

- Listener implementation = `log.info("*****Embedded subprocess has started");`
- Type = expression
- Event = start





Example 2:

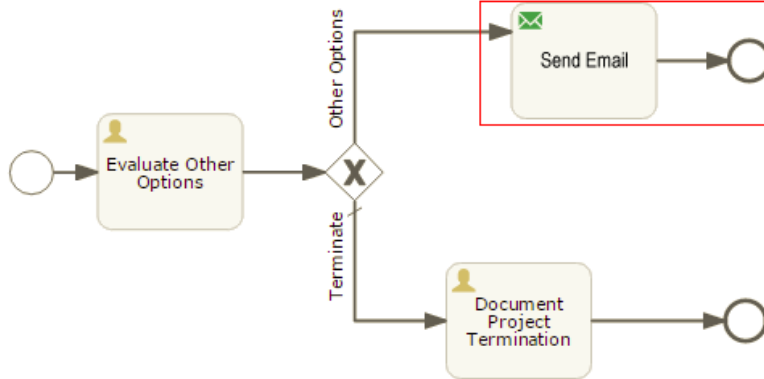
- Listener implementation = `${execution.setVariable("taskAssignee", task.assignee)}`
- Type = expression
- Event = complete

Example 3:

- Listener implementation = `${task.setName("Development and testing completed")}`
- Type = expression
- Event = create

Add task and execution listeners

1. Select **Main Menu**  > **Administration** > **Application Setup** > **Business Process Setup** > **Process Models**.
2. Select the model and click **Edit** .
3. Click **Visual Editor** .
4. Select the appropriate user task, for example, `Evaluate Other Options`
5. In **Properties**, select **Task Listeners**.
 - a. Select **Add listener** .
 - b. In **Event**, select **complete**.
 - c. Enter an **Expression**, for example: `${execution.setVariable("taskAssignee", task.assignee)}`
 - d. Click **Save**.
6. From **Activities**, click and drag a **Mail Task** to the subprocess.
7. Connect the sequence flow from the exclusive gateway to the mail task, then connect the mail task to an end event.



8. Select the **Mail Task**, then in **Properties**, do the following:
 - a. Enter a **Name**, for example: **Send Email**
 - b. Select **To** and enter the email address.
 - c. Click **Save**.
 - d. Select **HTML** and enter the code for the email message.


 **Example:**


```

<table>
  <tr>
    <td>Other Options:</td>
    <td>${otheroptiondescription}</td>
  </tr>
  <tr>
    <td>Submitted By (User Id) :</td>
    <td>${taskAssignee}</td>
  </tr>
</table>

```

- e. Click **Save**.
- f. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

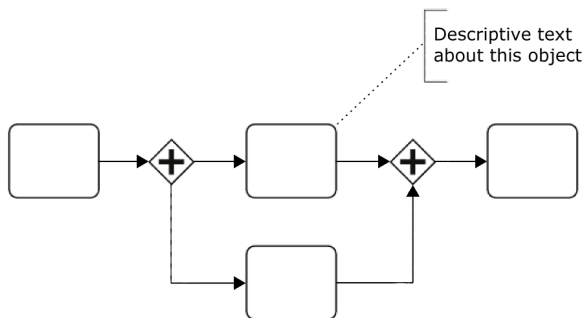
- g. Click **Save the Model** . Verify the settings, then click **Save & Close** or **Save**.

Chapter 7: Artifacts

Artifacts are comments about elements in a process. Comments do not affect the process.

Text annotation

Provides definitions or short descriptions



Connect an artifact to the relevant objects by an association line. The line can have a label.

- **Association**
-> **Directional association**
- <.....> **Bidirectional association**

Chapter 8: APIs

The following custom-made Activiti Service tasks are used to execute the public REST APIs. These were created in-house and are shipped with Workflow Designer.

- PersonalityMultiReadAPI
- DelegateAuthroityAPI
- DelegateProfileAPI
- DelegatorAPI
- RoleProfileAPI
- AttestationWorkflowAttributesAPI
- ResumeAPI
- Generic Kronos API
- GenericNotificationNotifyAPI
- IHubAPI
- RequestAPI
- RequestStatusChaneAPI
- RequestApproveRejectAPI
- AccrualsAPI

Chapter 9: Create Forms

Business Processes – in particular for [user tasks](#) – can require people to complete forms that collect or present data.

You create these forms in the Form Designer by dragging and dropping elements such as text fields, tables, check boxes, buttons, and so forth from a palette onto the Form Designer. Forms can also access data by way of REST endpoints from an outside process.

Note: Configure the pop-up blocker settings in your browser to allow pop-up windows. Otherwise, the Process Model designer does **not** open.

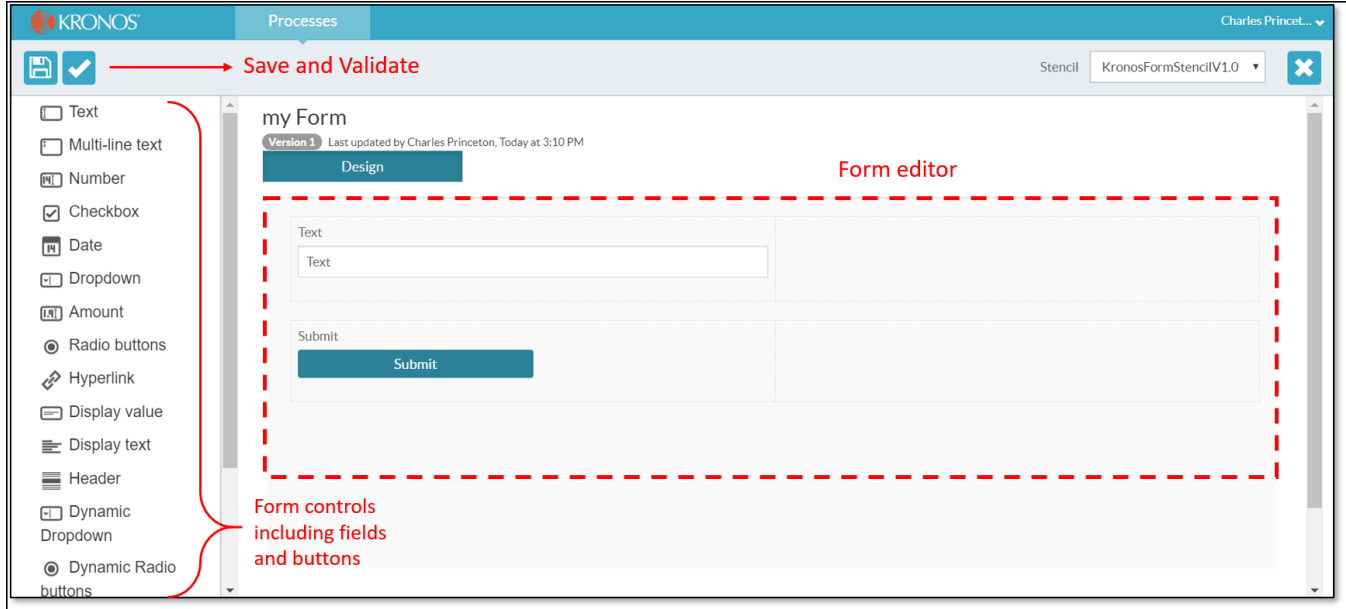
To start a form:

1. Drag a User task from the Workflow Designer Stencil set to the canvas.
2. From the **Property** panel at the bottom, select **Referenced form**.
3. Select **New form**.
4. In the Create a new form box, enter the **Form name** and **Description**.


Note: The **ADPFormStencil** is listed by default in the Stencil field. You cannot change this.

5. Click **Create form**.


The Form Designer opens.



The main components of the Form Designer are:

- The **Save and Validate** button  at the top of the Form Designer is used to save your progress and check that your form is valid.
- The **stencil palette** or form controls along the left side of the Form Designer displays the components you use to add elements to the forms you design.
- The **form editor** is the canvas or workspace that you use to arrange the elements that define the form.

Form editor stencil pallet

The following describes each of the elements available from the stencil palette. When you drag an element to the Form Editor, click the edit icon  to open the Edit dialog box. This dialog box is organized by a number of tabs across the top. All elements include the following **General** tab as well as additional element-specific tabs.

General tab

- **Label** – Enter the label name of the text field
- **Override ID** – Select this checkbox to make the ID property editable
- **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.

- **Required** – Select this checkbox to make the user input mandatory for the text field.
- **Placeholder** – Enter any placeholder value. This is optional.
- **Variable Name for Label (Overrides Default Label)** – Enter an execution variable for the field label. By using this execution variable, the user of the workflow can dynamically control the value of field label.

Text

The text field allows user input. It contains the following tabs:

- **General tab**
- **Advanced tab**
 - **Min Length** – Enter the minimum length allowed for the text.
 - **Max Length** – Enter the maximum length allowed for the text. The maximum character length for input value is 3999 characters.
- **Style tab**
 - **Color** – Click the drop-down to select the label color.
 - **Label Font Size Weight** – Click the drop-down to select the label font.

Multi-line text

The Multi-line text field allows user to input data in multiple lines of text. It contains the following tabs:

- **General tab**
- **Advanced tab**
 - **Min Length** – Enter the minimum length allowed for the text.
 - **Max Length** – Enter the maximum length allowed for the text. The maximum character length for input value is 3999 characters.
- **Style tab**
 - **Color** – Click the drop-down to select the label color.
 - **Label Font Size Weight** – Click the drop-down to select the label font.

Number

The Number field allows the user to input numbers. It contains the following tabs:

- **General tab**
- **Advanced tab**
 - Min Length – Enter the minimum length allowed for the text.
 - Max Length – Enter the maximum length allowed for the text. The maximum character length for input value is 3999 characters.
- **Style tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Checkbox

The Checkbox field enables you to add a checkbox that accepts true-or-false input. It contains the following tabs:

- **General tab**
- **Advanced tab**
 - Min Length – Enter the minimum length allowed for the text.
 - Max Length – Enter the maximum length allowed for the text. The maximum character length for input value is 3999 characters.
- **Style tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Date

The Date field allows the user to enter a date using a date calendar. It contains the following tabs:

- **General tab**
- **Advanced tab**
 - Minimum Date (d-M-YYYY) – Adds static minimum date restriction on user input.
 - Maximum Date (d-M-YYYY) – Adds static maximum date restriction on user input.
 - Maximum Date Offset (Overrides Maximum Date) – Adds maximum number of days offset restriction on user input. Offset value can be -ve/+ve. Enter a negative number for a date before the current date or a positive number for a date after the current dates.

- Select variable for minimum date (d-M-YYYY) – This is a variable-controlled field. When provided, the calendar uses this value for the minimum date. If the “Minimum Date (d-M-YYYY)” field is provided, the value of this field will be ignored
- Select variable for maximum date (d-M-YYYY) – This is a variable-controlled field. When provided, the calendar uses this value for the maximum date. If the “Maximum Date (d-M-YYYY)” field is provided, the value of this field will be ignored
- **Style tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Dropdown

- **General tab**
 - **Default Label** – Enter the label name of the text field
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
 - **Required** – Select the checkbox to make user input mandatory
 - **Type ahead** – Use this option to filter values displayed in the dropdown list as the user types in the dropdown field. The system filters the dropdown list to display only those values that contain the characters entered in the field. The filter is case-insensitive.
 - **Placeholder** – (optional) Any placeholder value.
 - **Variable Name for Label (Overrides Default Label)** – The execution variable name that contains the value for the field label.
- **Style Tab**
 - **Color** – Click the drop-down to select the label color.
 - **Label Font Size Weight** – Click the drop-down to select the label font.

Amount

- **General tab**
- **Advanced tab**
 - **Min Length** – Enter the minimum length allowed for the text.
 - **Max Length** – Enter the maximum length allowed for the text. The maximum allowed is 3999 characters.

- **Style tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Radio buttons

You can add multiple radio buttons, but select any one that should be the default.

- **General tab**
- **Options tab**
 - Options type – Manual
 - Options – Enter the name and ID of the options. If you need more than 2 options, click **Add a new option**.
- **Style tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Hyperlink

- **General tab**
 - Label – Enter the name of the field.
 - Override ID? – Select the checkbox to make the ID property editable.
 - ID – Enter the ID of the field control, which is also the process variable used to store field data.
- **URL options tab**
 - Hyperlink URL – Enter the URL here.
 - Form field or Variable – Enter the URL using the process variable or a field value.
 - Display text – Add the name that displays the link URL.

Display value

The Display value field is used for the value of any process variable or other field. It contains the following tabs:

- **General tab**
 - Default Label – Enter the name of the field.
 - Override ID? – Select the checkbox to make the ID property editable.
 - ID – Enter the ID of the field control, which is also the process variable used to store field data.
 - Form field or Variable – Select the existing form field or execution variable whose value will be displayed using the **Display value** field.
- **Style tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Display text

The Display text field is used to display pre-entered text data. It can also be used to display value of process variables and form fields. It contains the following tabs:

- **General tab**
 - **Label** – Enter the label name of the text field
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
 - **Text to Display** – Enter the text and the execution variable to be displayed..
 - **Form field or Variable** – Select the field ID or process variable for the data to be displayed.
- **Style Tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Header

- **General tab**
 - **Default Label** – Enter the label name of the text field
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
- **Style Tab**
 - Color – Click the drop-down to select the label color.
 - Label Font Size Weight – Click the drop-down to select the label font.

Dynamic Dropdown

- **General tab**
 - **Label** – Enter the label name of the text field
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
 - **Required** – Select the checkbox to make user input mandatory
 - **Type ahead** – Use this option to filter values displayed in the dropdown list as the user types in the dropdown field. The system filters the dropdown list to display only those values that contain the characters entered in the field. The filter is case-insensitive.
 - **Variable name for Dropdown Options** – Enter the variable name that carries the array object or list of items
 - **Variable Name for Label (Overrides Default Label)** – The execution variable name that contains the value for the field label.
- **Style Tab**
 - **Color** – Click the drop-down to select the label color.
 - **Label Font Size Weight** – Click the drop-down to select the label font.

Dynamic Radio buttons

- **General tab**
 - **Label** – Enter the label name of the text field
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
 - **Required** – Select the checkbox to make user input mandatory
 - **Variable name for Radio button Options** – Enter the variable name that carries the array object or list of items
 - **Variable Name for Label (Overrides Default Label)** –
- **Style Tab**
 - **Color** – Click the drop-down to select the label color.
 - **Label Font Size Weight** – Click the drop-down to select the label font.

Dynamic Check boxes

- **General tab**
 - **Label** – Enter the label name of the text field
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
 - **Required** – Select the checkbox to make user input mandatory
 - **Variable Name for Check Box Options** – Enter the variable name that carries the array object or list of items
 - **Variable Name for default selected Check Boxes** –
 - **Variable Name for Label (Overrides Default Label)** –
- **Style Tab**
 - **Color** – Click the drop-down to select the label color.
 - **Label Font Size Weight** – Click the drop-down to select the label font.

Submit, Next, Cancel, and Terminate buttons

Each of these buttons have the following attributes:


- **General tab**
 - **Default Label** – Enter the label name of the text field
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
- **Style Tab**
 - **Button Style** – Select Primary or Secondary.

Show My Timecard

You add this control to a form to redirect to the user's timecard rather than displaying the form.

- **General tab**
 - **Override ID?** – Select this checkbox to make the ID property editable
 - **ID** – Enter the ID of the text field, which is also the process variable used to store text field data.
- **Advanced tab**
 - **Variable name for timecard options**

Example: Create a form

Depending on the form you are designing, drag the required form elements to the canvas, then select each element one by one and click **Edit**  to define the parameters of the element.

1. On the **General** tab, provide the following information:

Default Label – Label name of the field.

Override ID? – The ID defaults to the default label name. To change this, select this check box and enter the new ID name in the **ID** field.

ID – The ID of the field control, which is also the process variable used to store field data.

Required – Select this check box if the user must enter information for this element.

Placeholder – Any placeholder value (optional). This will appear the first time

Variable Name for Label (Overrides Default Label) – Enter the variable name that contains a dynamic label. If you add a variable here, it overrides the **Default Label**. Note that the system does not validate this variable. See [Initialize Variables task on page 59](#) for information about defining variables.

Note: You cannot define a variable name for the label in the **Header, Dynamic Dropdown, Submit Button, Next Button, and Cancel Button** form elements.

2. If you are designing a **Text, Multi-Line Text, Number, Date, or Amount** form, the **Advanced** tab enables you to provide input validations by providing the following information:

- For number-based form elements (**Number** or **Amount**):

Minimum Value – Enter the minimum value that the user must enter.

Maximum Value – Enter the maximum that the user can enter.

- For text-based form elements (**Text** and **Multi-Line Text**):

Minimum Length (Maximum Length limit is 3999. No minimum limit. – Enter the minimum number of characters that the user can enter.

Maximum Length (Maximum Length limit is 3999. No minimum limit. – Enter the maximum number of characters that the user can enter.

- For the **Date** field, you can enter static dates or offset dates.

Static dates enable you to restrict the date between two dates:

- **Minimum Date (d-M-YYYY)**

- **Maximum Date (d-M-YYYY)**

Offset dates enable you to override the minimum and maximum dates and to set a date relative to the current date. Enter a negative number for a date before the current date or a positive number for a date after the current date. If you enter -20, for example, the date must be 20 days before the current date and if you enter +30, the date must be 30 days after the current date.

- **Minimum Date Offset**
- **Maximum Date Offset**

3. On the **Style** tab, optionally specify rich text format (RTF) attributes for form labels, headers, and button styles:
 - Labels – You can specify color and font size-weight. Examples include red, green, and blue; and small, medium, and large, respectively.
 - Headers – You can specify color and font size. Examples include red, green, and blue; and H4 - small, H3 - medium, and H2 - large, respectively.
 - Buttons – You can specify whether the button should have primary or secondary styling

Note: The definition of the attributes available in these form elements is determined by the branding styles in effect in your organization.

4. Click **Close** and repeat for other elements.
5. (Optional) Select **Tabs** or **Outcomes** to add these elements.
6. At the end of the form, select and drag to add the action buttons: **Next** or **Submit** and **Cancel**.
7. When you finish:
 - a. Click **Validate the Model** . Correct any errors.

Note: Models that fail validation **cannot** be deployed.

- b. Click **Save the Model** , then verify the settings.
- c. Click **Save and Close Editor** or **Save**.

Chapter 10: Custom Beans in Business Process

A custom bean is a Spring Expression Language (SpEL) component that can be a utility or a call to get common information into a Business Process. SpEL queries and manipulates object graphs at run time. Syntax is similar to Unified EL but adds method invocation and string templating.

Preconfigured custom beans

Preconfigured custom beans include the following:


- **ResourceBean**: Get the full name of a resource to send notifications.
- **FormInputFormatter**:
 - **convertDate**: Convert the date into a specific format.
 - **createRecipientsList**: Return the list of recipients, each separated by a colon (:).

You can use these beans to create new beans.

Use of custom beans in an expression

You can derive the value for a form field, service task class field, or other field from a variable, expression, or a default value, and can use custom beans in an expression.

- **Syntax**: `${beanName.methodName(Method args)}`

 **Example**: `${formInputFormatter.createRecipientsList(DelegateList)}`

Custom beans for custom extensions

Bean expression	Purpose	Example usage
<code>\${employee.getManager(<personNumber of the employee>, execution)}</code>	Get the name of the manager of an employee.	<code>\${employee.getManager("10015", execution)}</code> <code>\${employee.getManager(initiator, execution)}</code>
<code>\${resource.getFullName(<personNumber of the resource>, execution)}</code>	Get the full name of a person.	<code>\${resource.getFullName(initiator, execution)}</code>
<code>\${formInputFormatter.convertDate(<Date>)}</code>	Convert a date from the format in Activiti™ into the format that the API requires: MM/DD/YYYY.	<code>\${formInputFormatter.convertDate(startDate)}</code>
<code>\${formInputFormatter.fetchJSONObjectData(<JSONArray>, keyName)}</code>	Loop through the JSON Objects in a JSON Array, and return the value against the KeyName that is found.	<code>\${formInputFormatter.fetchJSONObjectData(jsonArrayVariable, "testKey")}</code>

Chapter 11: Business Processes for Devices

Devices, such as terminals, can display timecard information from ADP Workforce Manager. QuickGlances display the information by way of Universal Device Manager (UDM).

Variables

Throughout an instance of a Business Process, the UDM broker maintains the following variables for processing.

- **initiator** – Identifies the user who started the Business Process. For UDM, this value is always `UDMUSER`.
- **ext_tenantId** – Internal ID for the tenant. ADP Workforce Manager maintains IDs for tenants that map to the tenant ID of UDM.
- **wfc_url** – The host URL as configured in UDM.
- **badgeNumber** – Passes the badge number
- **selectedId** – List of IDs in a form

Authenticate REST API calls as `UDMUSER` or as an user being accessed with proxy access.

Forms

Devices require simpler forms than other types of Business Processes. So the format of forms must respect the following specifications to be transformed into the correct format for the device. Each field is generated by Activiti™ as JSON code; you cannot control the format of this code.

Fields to collect data must respect the following constraints:

Form name

Must begin with **SmartViewForm**

Container

UDM allows only one container in a form. That container contains all fields for the form.

Field types and transformations

The UDM broker supports a subset of available fields. UDM translates these fields into the format that the device supports. If an unsupported field is included, UDM throws an error.

The following field types are supported:

Text – Transforms to various field types on the device depending on the ID of the field

- **Badge Number** – The ID matches `badgenumber`. The device displays the badge entry form for a 6-character badge number.
- **Time** – The ID with `time`. Examples: `time_meeting` or `time_lunch`. The device displays a field to enter time in the format `HH:MM`.
- **Duration** – The ID starts with `duration`. Examples: `duration_meeting` or `duration_lunch`. The device displays a field to enter a duration in hours and minutes.
- **String** – The ID does **not** meet any of the above criteria. The device displays a text field.

Number – The device displays a field to enter an integer value with no decimals.

Date – The device displays a field to enter a date in the format `YYYY-MM-DD`.

Dropdown – The device displays a list of items to select.

- Each option in the form is an item in the list.
- On the device, the person selects one item to advance the process.
- For each option, the ID value is the selected ID parameter value that is passed to the Business Process Model.
 - This value is set in the `selectedId` variable.
 - Typically, the instance of the Business Process evaluates this value for processing.

Examples:

Determine input to an API call.

 Determine in which direction to branch the instance of the Business Process.

Field Labels – For each field, the UDM broker uses the value set for **Label** as the field label on the device.

Form with select options

The ID of the selected item goes into the `selectedId` variable. In the Business Process Model, this type of form contains radio buttons. UDM translates this form for the device as a list of items from which the person selects one.

- The form must have only one container.
- This container contains the Radio buttons element.
- The form is translated into a form of selectable items on the device; each radio button becomes a selectable item.
 - The person selects one item from the device. Each selection advances the process.
 - The label in the form is the label on the device.
 - The ID value set for the radio option is the `selectedId` value that is submitted when selected.

Chapter 12: Integrations in a Business Process

Note: The ability to run integrations in a Business Process is disabled by default. To have it enabled, you must submit a support case.

The difference between Business Processes and integrations are:

- Business Processes include tasks that can be automated or done by people.
- Integrations process only data and require minimal or no human intervention.

A Business Process can run an integration from the data integration service task so long as the input parameters are shared. The task calls the integration by way of an API. The Name of the integration is the only input parameter; you cannot specify other parameters of the integration.

Caution: By default, a Business Process retrieves data by using the credentials of the person who initiates the process. However, a process may need to retrieve data that the initiator of the process cannot access. Do this **only** when absolutely necessary and be careful to **not** expose data to people who are not privileged to access that data. To access this data, make the API request on behalf of another person; a best practice is to create a user account specifically for this purpose rather than use another real person's credentials.

To pass integration-specific parameters, use one of the following:

- [Generic REST Connector](#) – Call any REST API.
- [Generic Upload API Task](#) – Attach a configuration file to create a service task to call any API.
- [API service tasks on page 44](#) – Sequence multiple API calls to run an integration so that the user has to enter parameters as part of the Business Process.

Run an integration or integration set

Endpoint URL	HTTP method	Publishable	Query	Description
http:// {{host}}/wfc/restcall/v1/platform/integration_ instances/execute	POST	yes	n/a	Create an integration instance and submit it for immediate execution in the system.
http:// {{host}}/wfc/restcall/v1/platform/integration_ set_instances/execute	POST	yes	n/a	Create an integration set instance and submit it for immediate execution in the system.

Run an integration example

```
{
  "id": 1,
  "name": "Factory job update",
  "description": "Job update for the main factory",
  "user": null,
  "requestName": null,
  "userParameters": [
    {
      "name": "disqualifyActive",
      "type": "RADIO",
      "value": true
    }
  ],
  "scheduledInfo": null,
  "integrationRequestId": null,
  "integrationType": "NONE"
}
```

Schedule an integration or integration set

The user has to enter schedule details such as frequency, start date, and so on.

Endpoint URL	HTTP method	Publishable	Query	Description
http:// {{host}}/wfc/restcall/v1/platform/integration_ instances/schedule	POST	yes	n/a	Create an integration instance and schedule it for execution in the system.
http:// {{host}}/wfc/restcall/v1/platform/integration_ set_instances/schedule	POST	yes	n/a	Create an integration set instance and schedule it for execution in the system.

Schedule an integration example

```
{
  "integrationDetails": {
    "id": 1,
    "name": "Factory job update",
    "description": "Scheduled job update for the main factory",
    "user": null,
    "requestName": "Factory job update-12-22-2017 11:44",
    "userParameters": [
      {
        "name": "disqualifyActive",
        "type": "RADIO",
        "value": true,
        "templateParamName": "disqualifyActive"
      }
    ],
    "scheduledInfo": "Recurs by day starting December 16",
    "integrationRequestId": null,
    "integrationType": "NONE"
  },
  "eventDetails": {
    "eventId": null,
    "restPost": null,
    "schedule": {
      "frequency": null,
      "frequencyType": 2,
      "interval": "4",
      "startTime": "12:12pm",

```

```
📄 "endTime": null,  
  "forever": true,  
  "startDateTime": "12/16/2017",  
  "endDateTime": null  
},  
"event": null  
}  
}
```

Chapter 13: Exception Handling

Business Processes must be able to handle exceptions, especially if the processes are long-running or cannot easily restart. Without exception handling, processes can fail without notifying anyone, or the behavior of a process can be unpredictable.

Types of exceptions:

- **Technology exceptions** – Example: Mail server is not available.
- **Business exceptions** – Example: Data is not valid.

[Toggle Examples](#)

Methods to handle exceptions

You can use the following methods to handle exceptions:

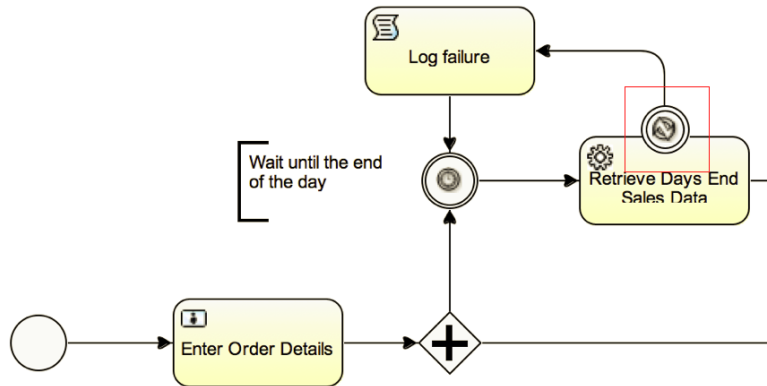
- [Boundary event error handler on page 131](#)
- [Event subprocess on page 132](#)
- [Scripts on page 133](#)
- [Transaction boundaries on page 134](#)
- [Compensation on page 134](#)

Boundary event error handler

The attributes of this type of error handler are:

- Simple
- Good for re-trying events and explicit control over the execution path
- Can result in complicated diagrams

Example exception handling boundary event

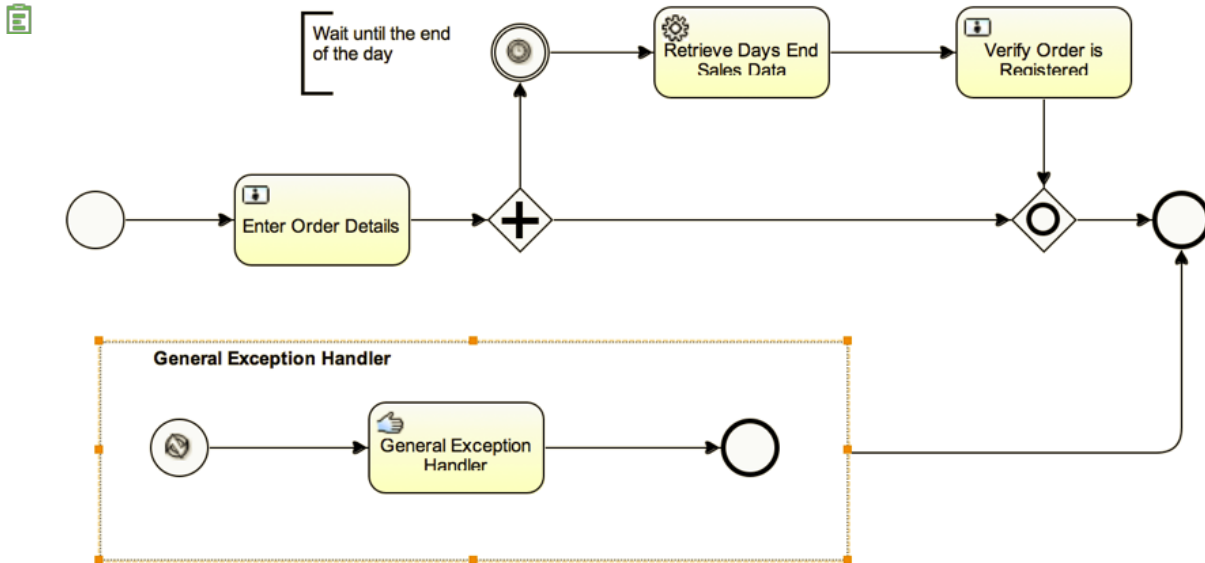


Event subprocess

Event subprocesses neatly model common exception-handling routines.

- Consistent handling of exceptions: All logic is in a single location and can be triggered from multiple steps.
- An error start or message start event triggers the subprocess which is a separate container that catches all events that have the same error code.
- The point of re-entry into the process is fixed.

Example exception handling subprocess



Scripts

The attributes of this type of error handler are:

- Throw exceptions from scripts in a try/catch block.
- The BPMNError can be thrown in JavaDelegates, scripts, expressions, and delegate expressions to trigger a boundary error event.
- The exception is caught and forwarded to the corresponding error handlers such as a boundary error event or an error event subprocess.

Example exception handling script

```

try {
    executeBusinessLogic();
} catch (BusinessException e) {
    throw new BpmnError("BusinessExceptionOccurred");
}
  
```

Transaction boundaries

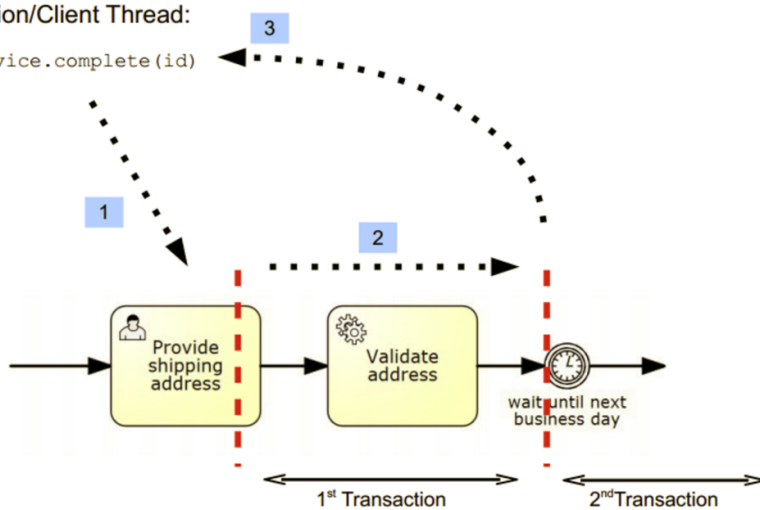
1. The client thread (Servlet) completes the task.
2. Because the service task is synchronous, validation occurs on the servlet thread. The intermediate timer event signals a wait state and commits the transaction.
3. If an error occurs, the entire transaction including the user task is rolled back.



Example exception handling transaction boundaries

Application/Client Thread:

```
taskService.complete(id)
```



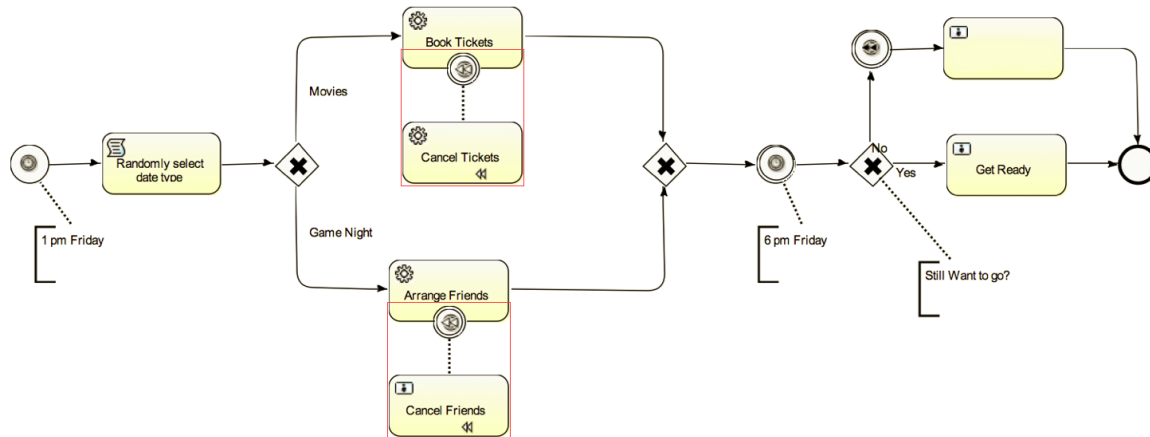
Compensation

The attributes of this type of error handler are:

- Compensation allows roll-back or cancellation of service tasks if a compensation event occurs during the current process or subprocess.
- The process or subprocess must trigger compensation, and the attached activity must successfully complete.
- Subprocesses that send a cancel event trigger compensation before the cancel flow starts.



Example exception handling compensation






Configure exception handling

This section includes the following ways to configure exception handling:

- [Error handling from a script task on page 135](#)
- [Error handling from a gateway on page 137](#)
- [Error handling from a Generic REST Connector or service task on page 141](#)


Error handling from a script task

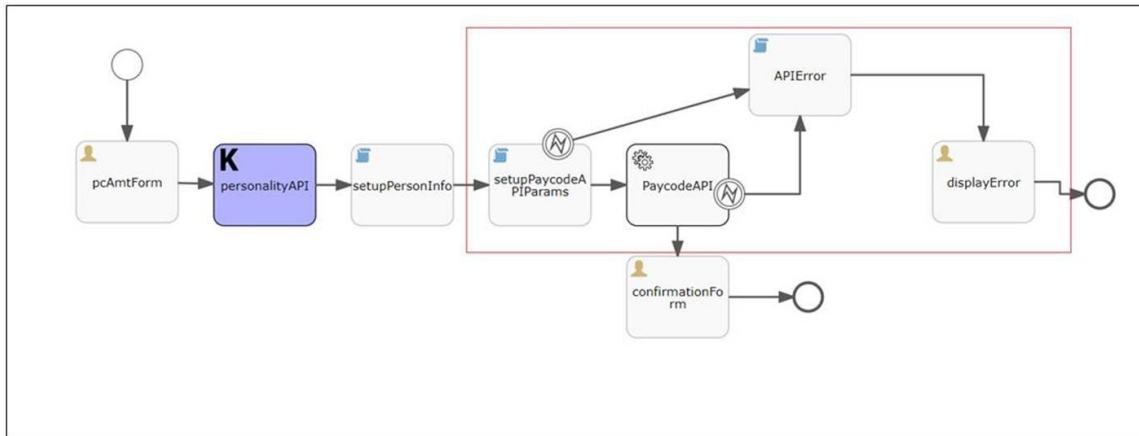
1. Click Main Menu  > Administration > Application Setup > Business Process Setup > Process Models.
2. Select the model. Click **Edit** .
3. Click **Visual Editor**  **Visual Editor**.
4. From the Stencil Set, click and drag a **Boundary Error Event** to the tasks in the process that can trigger an error.



Example error handling from a script task

Error handling tasks are within the red frame.

-  Add the boundary error event to the **setupPaycodeAPIParams** script task and the **PaycodeAPI** service task in the following example.



- From the Stencil Set, click and drag a **Script Task** to the appropriate place in the Process Model.
- Select the script task. In properties:
 - Enter a **Name**, for example: `APIError`
 - In **Script Format**, enter **groovy**.
 - Select **Script**.
 - Enter the following code:


```


def restErrorCode = execution.getVariable("WFD_ERROR_CODE")
def restErrorMsg = execution.getVariable("WFD_ERROR_MESSAGE")
restErrorMsg = restErrorMsg!=null ? restErrorMsg : "";
if(restErrorCode) {
String errorMsg = restErrorCode + " :: " + restErrorMsg;
execution.setVariable("errorMessage", errorMsg)
execution.setVariable("bpmErrorMessage", errorMsg)
} else {
def bpmError = execution.getVariable("BPMN_ERROR_MESSAGE")
String errorMsg = "API Error"
bpmError = (bpmError == null ? errorMsg : bpmError)
execution.setVariable("errorMessage", errorMsg)
execution.setVariable("bpmErrorMessage", bpmError)


```


}




7. Link the boundary error events to the new script task.
8. From the Stencil Set, click and drag a **User Task** and link it to the new script task.
9. Select the user task. In properties, enter a **Namesuch** as `displayError`
10. From the Stencil Set,, click and drag a **User Task** and link it to the last task in the non-error path.
Example: Link it to **PaycodeAPI**
11. Select the user task. In properties, enter a **Namesuch** as `confirmationForm`
12. Click **Save**.
13. **Verify the exception flow** – To test that exceptions flow to the subprocess, add a script task that throws an exception.
 - a. From **Activities**, select, drag, and connect a **Script Task** after the first user task.
 - b. Select the script task.
 - c. Enter a **Namesuch** as `Throw Exception Test`
 - d. Select **Script**.
 - e. Enter the Groovy script:

```
import org.activiti.engine.delegate.BpmnError;
throw new BpmnError("rejectedException");
```
 - f. Click **Save**.
 - g. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

- h. Click **Save the Model** , then verify the settings and click **Save and Close Editor** or **Save**.
- i. Deploy, run, and test the model.

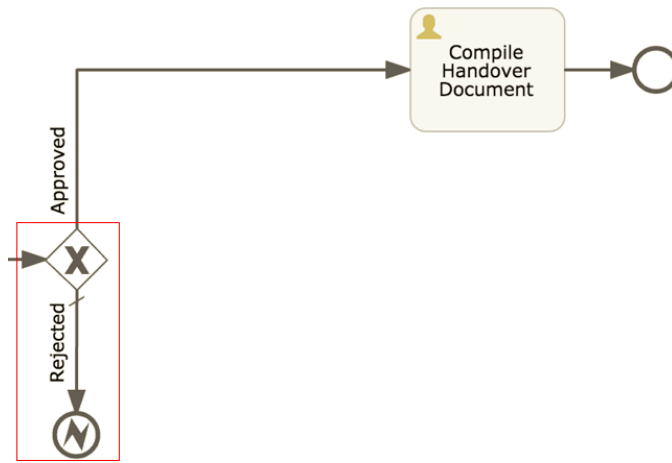
Error handling from a gateway

1. Click **Main Menu**  > **Administration** > **Application Setup** > **Business Process Setup** > **Process Models**.
2. Select the model. Click **Edit** .
3. Click **Visual Editor**  **Visual Editor**.

4. Throw exceptions

- From the Stencil Set, select **Gateways**, then click and drag an **Exclusive Gateway** to the Stencil Set.
- From **End Events**, click and drag a **Error End Event** to the subprocess.
- Connect the Rejected sequence flow to the error end event.

Example:

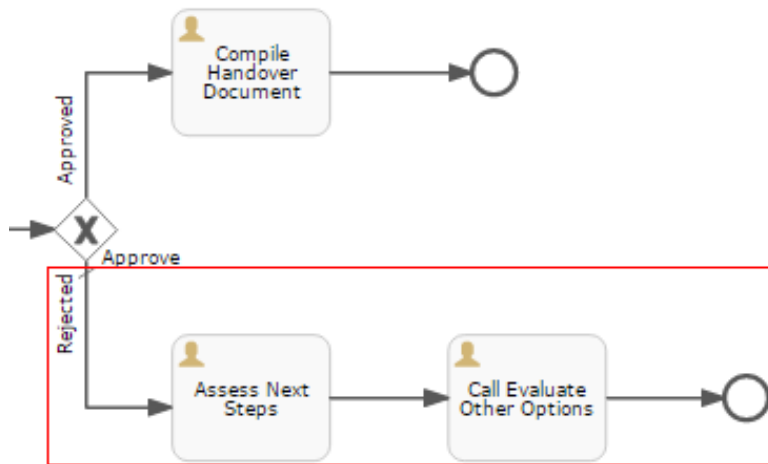


- Select the error end event.
- In **Parameters > Error Reference**, enter `rejectedException`.

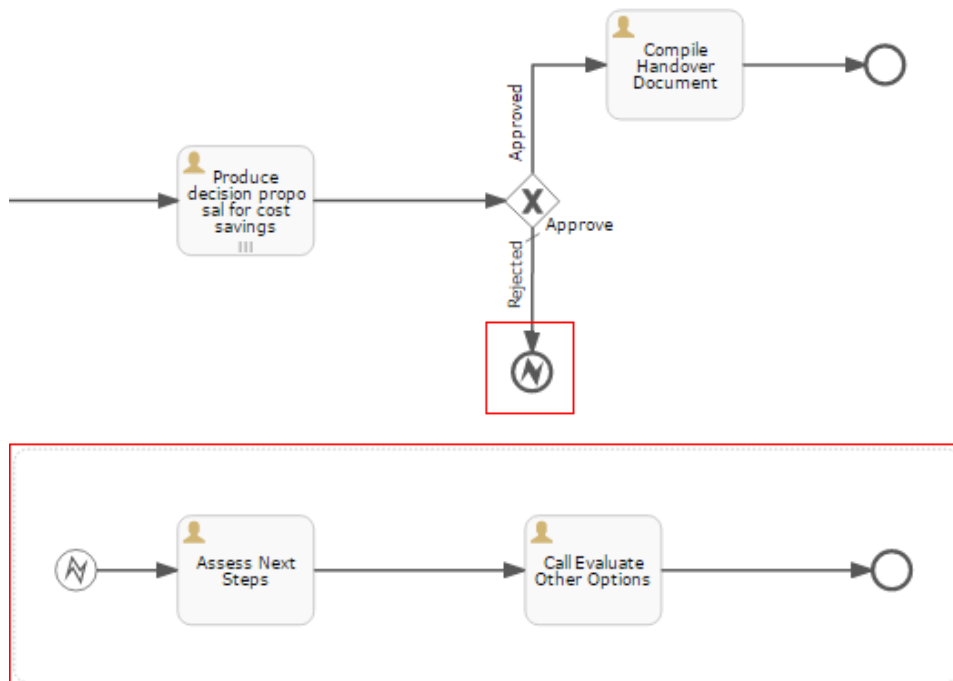
5. Handle exceptions with an event subprocess

- Select the appropriate user tasks such as `Estimate project costs`
- From **Structural**, click and drag a **Subprocess** to the canvas.
- From **Start Events**, click and drag a **Start Error Event** to the subprocess.
- Select the start error event.
- In **Properties > Error Reference**, enter `rejectedException`. This binds the subprocess to the error end event that you added.
- Move the tasks from the Rejected path of the exclusive gateway into the new subprocess.


Example – Before




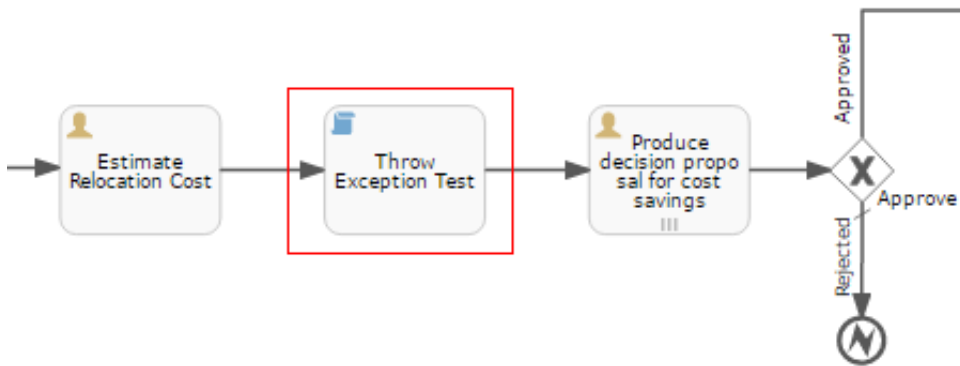
Example – After




- g. Click **Validate the Model** . Correct any errors.


 **Caution:** Models that fail validation **cannot** be deployed.


- h. Click **Save the Model** , then verify the settings and click **Save and Close Editor** or **Save**.
 - i. Deploy, run, and test the model.
6. **Verify the exception flow.** To test that exceptions flow to the subprocess, add a script task that throws an exception.
- a. From the Stencil Set, select **Activities**, then select, drag, and connect a **Script Task** after the first user task, for example:



- b. Select the script task.
- c. Enter a **Name**. Example: `Throw Exception Test`
- d. Select **Script**.
- e. Enter the Groovy script:

```
import org.activiti.engine.delegate.BpmnError;
throw new BpmnError("rejectedException");
```
- f. Click **Save**.
- g. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

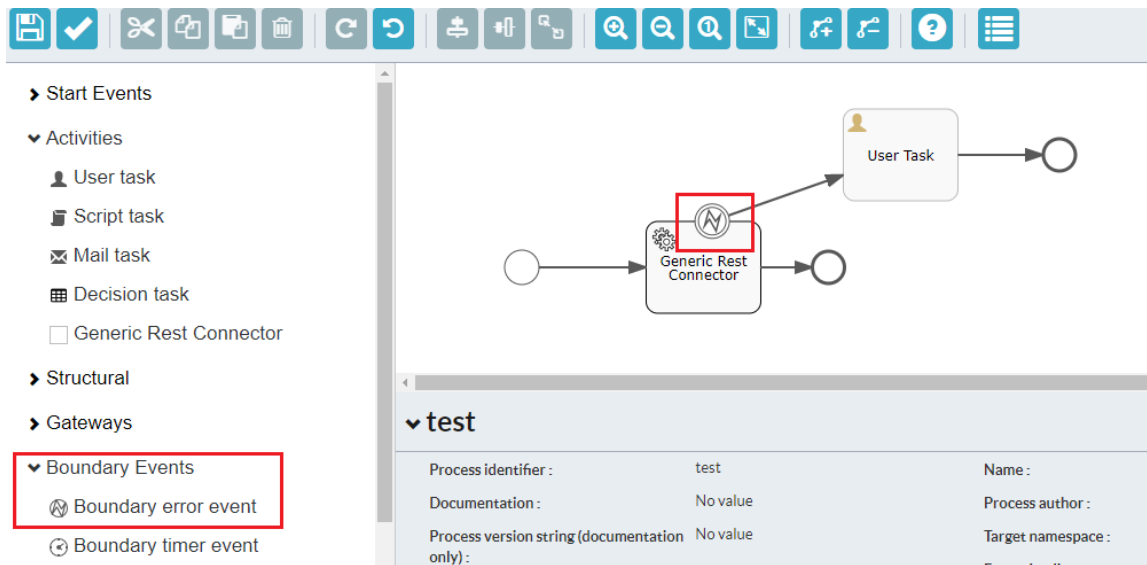
- h. Click **Save the Model** , then verify the settings and click **Save and Close Editor** or **Save**.
- i. Deploy, run, and test the model.




Error handling from a Generic REST Connector or service task

To handle errors in the [Generic REST connector](#) or other service tasks, use a boundary error event that catches a process error (BPMNError), interrupts the task, and jumps the process to an error start or error catching event.

This way, the process does not break because of an error.


Example error handling in a service task





1. Click **Main Menu**  > **Administration** > **Application Setup** > **Business Process Setup** > **Process Models**.
2. Select the model. Click **Edit** .
3. Click **Visual Editor**  **Visual Editor**.
4. From the Stencil Set, select **Boundary Events**.
5. Click and drag a **Boundary Error Event** to the service task in the process.


When an error is thrown by the service task or REST connector, the **BPMN_ERROR_MESSAGE** execution variable sets the error message.

- Use this variable in a script task as `execution.getVariable("BPMN_ERROR_MESSAGE")`
- Also you can use this variable directly in forms as `${BPMN_ERROR_MESSAGE}`.

 **Note:** Do not provide the error reference in the boundary error event because this reference is used only when the type of error is known. The error reference is controlled in the code.

6. Add a new task or path from this event that routes the process to a script or user task if an error occurs. You can use `BPMN_ERROR_MESSAGE` or a custom message in this path for errors.
7. Click **Validate the Model** . Correct any errors.

 **Caution:** Models that fail validation **cannot** be deployed.

8. Click **Save the Model** , then verify the settings and click **Save and Close Editor** or **Save**.
9. Deploy, run, and test the model.

Chapter 14: Business Process Errors

The error messages for most Business Process errors provide a solution. The following errors, however, can be the result of a number of different conditions.

Error Message	Issue	Probable Causes
Business Process Engine is unavailable. You cannot deploy a process model at this time.	The Workflow Engine is down.	<ul style="list-style-type: none"> • ILB is down. • NGINX is down. • Webagent is down. • Tomcat server is down. • Database is down.
Business Process Designer is unavailable. You cannot create, edit, or delete a process model at this time.	Workflow Designer is down. (The Workflow Engine and Workflow Designer are on the same server.)	<ul style="list-style-type: none"> • ILB is down. • NGINX is down. • Webagent is down. • Tomcat server is down. • Database is down.
Business Process Engine is unauthorized. You cannot deploy a process model at this time.	The tenant is not present in the Workflow Engine.	<ul style="list-style-type: none"> • The workflow service failed to create the tenant during tenant provisioning. • The tenant was manually deleted from the workflow service. • The user could not be created on the workflow service.
Business Process Designer is unauthorized. You cannot create or edit a process model at this time.	The tenant is not present in the Workflow Designer.	<ul style="list-style-type: none"> • The workflow service failed to create the tenant during tenant provisioning. • The tenant was manually deleted from the workflow service. • The user could not be created on the workflow service.
This Process Model cannot be deployed. Refer to the online help	The system encountered an error while deploying a model.	<p>A Process Model will fail to deploy for the following reasons:</p> <ul style="list-style-type: none"> • Any internal server error occurred.

Error Message	Issue	Probable Causes
for more information.		<ul style="list-style-type: none"> The process contains a script task and the script contains restricted coding patterns such as: <ul style="list-style-type: none"> import groovy.sql import java.sql import com.gmongo Sql.newInstance jdbc:edb jdbc:mysql jdbc:db2 jdbc:oracle jdbc:postgresql jdbc:hsqldb jdbc:sqlserver jdbc:jtds MongoCredential.create GroovyShell
The Business Process cannot be transferred. Refer to the online help for more information.	The system encountered an error while transferring a process model through SDM.	<p>A Process Model will fail to deploy for the following reasons:</p> <ul style="list-style-type: none"> Any internal server error occurred. The process contains a script task and the script contains restricted coding patterns such as: <ul style="list-style-type: none"> import groovy.sql import java.sql import com.gmongo Sql.newInstance jdbc:edb sql.execute sql.rows jdbc:edb jdbc:mysql jdbc:db2 jdbc:oracle jdbc:postgresql jdbc:hsqldb jdbc:sqlserver jdbc:jtds MongoCredential.create GroovyShell

Error Message	Issue	Probable Causes
<p>This task has failed runtime validation and cannot be completed. Refer to the online help for more information</p>	<p>The system failed runtime validation while executing the workflow from Control Center.</p>	<ul style="list-style-type: none"> • Rest API call failure in Internal Rest connector, Generic Rest connector, and Service tasks. • No form attached to user task. • Missing error handling • Any design error that was not validated during the Process Model design time, such as missing evaluation of sequence flow • Any expression that could not be evaluated • Script task contains restricted coding patterns such as: <ul style="list-style-type: none"> import groovy.sql import java.sql import com.gmongo Sql.newInstance jdbc:edb jdbc:mysql jdbc:db2 jdbc:oracle jdbc:postgresql jdbc:hsqldb jdbc:sqlserver jdbc:jtds MongoCredential.create GroovyShell
<p>Service unavailable</p>	<p>Workflow service while executing the Business Process from Control Center.</p>	<p>The Workflow service is either unavailable or the user is not authorized to access it.</p>

Chapter 15: Troubleshoot Business Processes

You can troubleshoot Business Processes using the Business Process Admin and the ADP Workforce Manager Business Processes error log.

Use the Business Process Admin

Use the Business Processes Admin to monitor and troubleshoot Business Processes.

1. From the Main Menu, select **Administration > Business Processes**.

The Business Process Admin is displayed in a new tab.

2. Select any of the following tabs:
 - **Deployments** – Lists application deployments by tenant and version. You can redeploy or delete applications.
 - **Definitions** – Lists deployed process definitions. You can sort or filter definitions by tenant, category, or name.
 - **Instances** – Lists current running process instances. You can complete a running instance.
 - **Tasks** – Lists current tasks. You can reassign or complete tasks.
 - **Jobs** – Lists current jobs. Jobs can be manually completed.
3. Expand results to display more detailed information about them.

Log Business Processes in ADP Workforce Manager

Use logs to troubleshoot errors encountered when running Business Processes.

About logs and logging

You can run logs for multiple Business Processes at the same time; consider limiting concurrent logs to 5. The Business Processes page shows the time the log was created, and the Log Status column indicates whether the log is running or has been stopped.

Logs run for a maximum of 30 minutes but can be stopped manually when you wish. See [Stop logging on page 149](#) for more information. Logs are saved for 48 hours before they are purged from the system. Consider exporting any logs you wish to save. See [Export a log on page 150](#) for more information.

If you stop logging and then start logging the same Business Process, the system creates a new log; the previous log is overwritten and cannot be retrieved. The system does not merge logs for the same Business Process.

Note: Business Processes logging might be disabled in some production environments.

Logged information

Business Processes logs contain runtime errors extracted from the stack trace that is generated when you start the log. Logs present this information:

- **Process Instance ID**— The unique instance identifier of the Business Process run in which the error occurred.
- **Component** — The name of the stencil set item in the Business Process that caused the error.

Note: Stencil set items without a name appear as "—none—" in the log. To make troubleshooting more efficient, give stencil set items meaningful names when adding them to your model.

- **Error Description** — The error message extracted from the stack trace that is run when you start the log. Error messages are truncated at 512 characters.
- **Log Time** — The time the error was captured.

Start logging

You start a log on a running Business Process.

To start logging:

1. From the Main Menu, navigate to **Administration > Application Setup > Business Process Setup > Business Processes**.
2. From the Process Models page, select the Business Process you want to start logging.

3. Click the **Start Log** button.

The Log Creation Date column displays the date and time the log was started. Time is displayed using a 24-hour clock.

The Log Status column displays Running.

Note:
You might have to refresh your browser to update the Log Status column.

Stop logging

Note: The system stops logs after they have been running for 30 minutes. You can stop them manually at any time.

To stop logging:

1. From the Main Menu, navigate to **Administration > Application Setup > Business Process Setup > Business Processes**.
2. From the Process Models page, select the Business Process whose log you want to stop.
3. Click the **Stop Log** button.
The Log Creation Date column retains the time stamp from when the log was started.

The Log Status column displays Inactive.

Note:
You might have to refresh your browser to update the status of the Log Status column.

Display a log

To display a Business Processes log:

1. From the Main Menu, navigate to **Administration > Application Setup > Business Process Setup > Business Processes**.
2. From the Process Models page, select the Business Process whose log you want to view.
3. Click the **View Log** button.
The log is displayed in a new browser window.

**Note:**

If no errors were logged, the log displays this message:

`This table currently contains no data.`

Empty logs cannot be exported.

Export a log

The Business Processes log is exported as a CSV file to your browser's default download location. Error logs have file names in this format:

`Process_Error_Log_<Process Definition Id>.csv`



Note: Before you begin, ensure that pop-ups are enabled for your browser.

To export a Business Processes log:

1. View the log as described in [Display a log on page 149](#).
2. Click the **Export** button. (This button is disabled if the log is empty.)
The log is exported to your browser's default download location.