

I. Introduction

This tutorial is intended to show the usage of the APEX extension *Flows for APEX 22.2* as well as explaining the integration of a flow into an APEX application. It also incorporates some good practices. After this tutorial and with some training/coaching, you are ready to build your own process-based applications!

We assume that you have basic understanding about Oracle APEX. If not, we suggest you follow the steps outlined in this blog post: <http://nielsdebr.blogspot.com/2019/08/getting-up-to-speed-with-oracle.html>

Also, this tutorial is not meant to explain the BPMN 2.0 standard. There are good books and online materials already available explaining this.

For further information about Flows for APEX and documentation you may visit the website: <https://flowsforapex.org/>

A minimum version of APEX 22.1 is required to be able to work through all the exercises in this tutorial. If you use a free workspace on apex.oracle.com, you will get the most current version of APEX directly. Even if Flows for APEX supports down to APEX 20.1, note that working on an older version of APEX might cause unpredictable issues and is strongly discouraged for this tutorial.

II. Contents

Exercise No.	Abstract
Exercise #01: Installation of Flows for APEX	Install the newest version of Flows for APEX inside your APEX workspace
Exercise #02: Model your first flow	Model an example flow with the Flows for APEX bpmn modeler to define a simple business process
Exercise #03: Create data model	Create the underlying SQL tables for storing the explicit process data
Exercise #04: Create application	Create the application with all pages, reports, and forms, that are needed within your business process
Exercise #05: Link application to flow	Link the application elements to your flow to have the process steps controlled by your bpmn model
Exercise #06: Define user roles	Define the APEX user roles and authorization schemes
Exercise #07: Testing	Test your application by creating test cases and observe the process using the Flows for APEX monitoring plugin
Exercise #08: Versioning	Deal with changes and multiple versions of your flow
Exercise #09: Process Variables	Use process variables on different places in your business process to enhance working with your flow
Exercise #10: Error handling	Detect and handle errors in your flow by using the Flow Monitor tools
Exercise #11: Email templates	Send emails from within your flow using templates.
Exercise #12: APEX Approval Task	Use the new APEX Approval Task Component natively in your Flows for APEX process
Exercise #13: Call Activities	Modularize your diagrams by nesting your business processes using call activities

1. Exercise #01: Installation of Flows for APEX

If not already available, apply for a free APEX workspace on apex.oracle.com.

Download the newest version of Flows for APEX from <https://flowsforapex.org/>

Install Flows for APEX by importing the file

FLOWSFORAPEX_EN_<apex-Version>_<application_id>_UTF8_<version>.sql

from the *Applications* folder into your APEX workspace. During the installation process select the default options including:

- choose *UTF.8* as file character set
- select *Auto Assign New Application ID (default)*
- install the *supporting objects*

Additionally, to enable the timer functionality for your flow, you need to add a certain DBMS Scheduler job to your workspace. Install it by importing and executing the file **enable_timers.sql** from inside the folder *Enable_Timers* using the **SQL Scripts** function of the **SQL Workshop**.

Notice that you might not have the right to create new jobs inside your own private workspace. In this case, the previous execution of the SQL statement “grant create job to <workspace_schema>”, invoked by the workspace sys, is necessary. Inside your free APEX workspace at apex.oracle.com, this right is already granted.

If you want, you can install the Sample Process Flow Application on the same way by importing the file

FLOWSFORAPEX<version>_SAMPLE_APP_EN_<apex-Version>_<application_id>_UTF8.sql

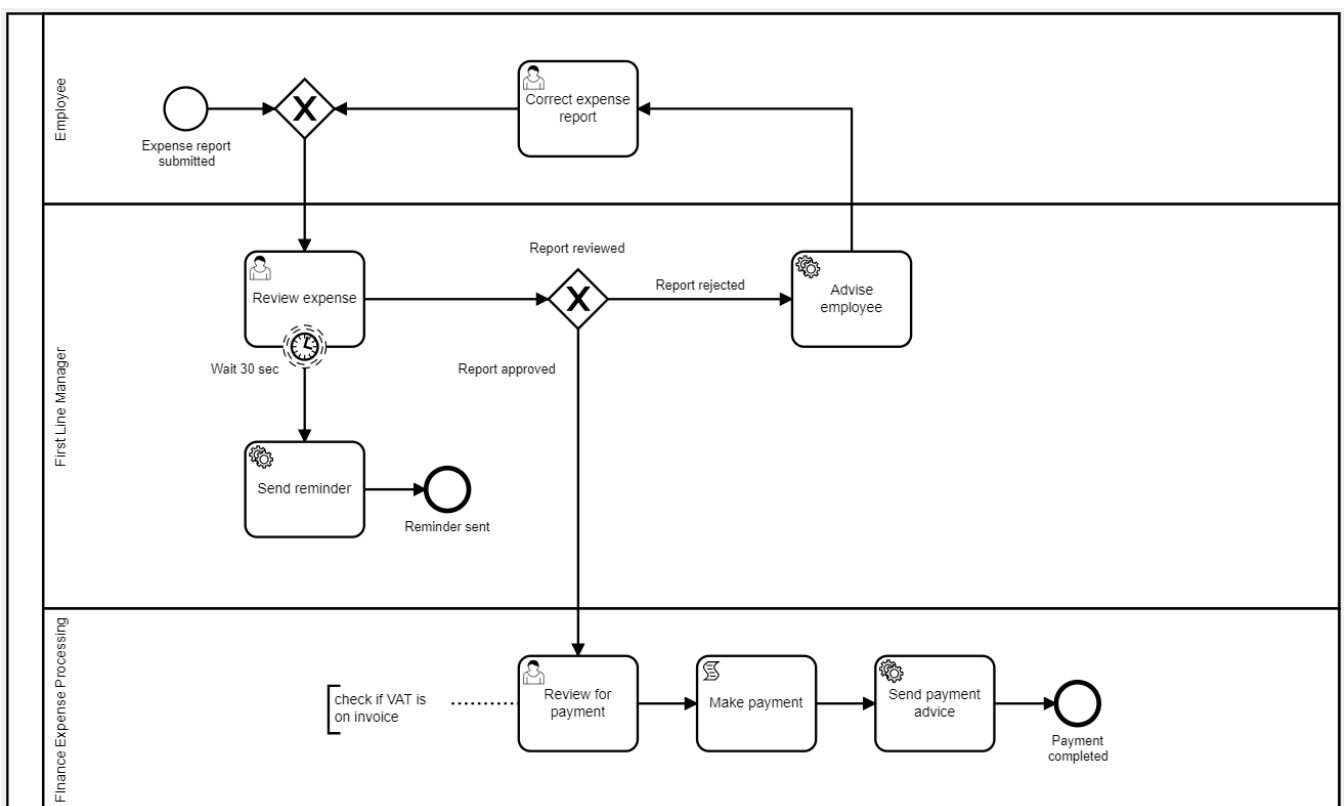
from the download folder. This app provides a demonstration for integrating a modelled flow into an APEX application.

2. Exercise #02: Model your first flow

Start the previously installed *Flows for APEX* application. Switch to the **Flow Management** page (second entry in the navigation menu) and click the **Create Model** button at the top right.

Enter *ExpenseFlow* as **Name** and *0* as **Version**. If you want, you can specify a **Category**, to help organizing your different flows inside your workspace. Create your new flow by clicking the button **Create**. At the top of the page, you can see the attributes of the current process as well as stats about the process instances. Click the **Modify Diagram** button below to open the flow modeller tool.

Model the depicted BPMN process by combining the specific elements as shown below. Also observe the following hints:



- All BPMN elements can be accessed via the toolbar on the left. Click the needed element and position it on the modelling area. After placing an element, you can access the different types by clicking the **wrench** icon displayed next to it after selection. In this example you need *User*, *Service* and *Script* tasks. Apart from that, this flow contains *Start*, *Boundary* and *End Events*, *Exclusive Gateways* and *Swim Lanes* (modelled by using the **Create Pool/Participant** function on the toolbar and dividing it afterwards using the corresponding button right to it).
- After selecting a placed element, additional settings are shown in a separate tab at the right. For this tutorial, set each element's ID to its name in underscore notation (e.g., Submit expense report -> Submit_expense_report). In general, IDs should be self-explanatory identifiers that presumably won't change during development to prevent errors when using IDs outside of the flow. You can also add the object type (Task, Gateway, Event) as a prefix to all IDs to simplify identification of different

objects. Proceed for all tasks and events as well as the gateway routes. For non-labelled sequence or message flows or collaborations, you do not have to assign a specific ID. Implementation details such as the *30 sec* on the timer event shouldn't be used inside the ID. Use something like *Event_wait_for_reminder* here instead.

- To enable role authorization for multiple lanes in your flow, the lane IDs must match the corresponding user roles static IDs. Since they consist of capital letters, use capital letters for the IDs as well or convert them to uppercase in your views (as done later in Exercise 03). *Note: for large flows, you might want to avoid the usage of lanes to preserve readability.*
- The shown Timer Event can be modelled by dragging a boundary event to the border of a task and changing its type to *Timer Boundary Event (non-interrupting)* afterwards by clicking the **wrench** icon. In the properties panel, set the **Timer Definition Type** to *Duration (Oracle)* and enter *000 00:00:30* in the second **Timer Duration** field to define a 30 sec duration. Alternatively, you can use the ISO 8601 syntax by selecting *Duration (ISO 8601)* as **Timer Definition Type** and *PT30S* for the **Timer Definition**. The source of the depicted arrow pointing to the *Send Reminder* task must be set to that Timer Boundary Event.
- To run timer events and resulting actions in the called subprocess properly, a valid APEX session is needed. If no session is available when the timer fires, the Flows for APEX engine will create a session based on predefined parameters. You can define these parameters on process level by clicking the **participant** object (the container for the three lanes) and specifying the *Background Task Session* attributes there. In this case you can select the Flows for APEX application or your own app, which you will create later and use any username in the environment. In business cases, this information might be given by process variables instead, to ensure, that the session can be linked to a certain business case.
- The **Text Annotation** as show at the *Review for payment* task can be used, to give extensive information about elements in the process. Alternatively, you can use the **Element Documentation** field inside the element's settings tab to provide a reference, for example a documentation page. All those documentation elements have no further usage inside an end user application depending on that flow.

When done with modelling, click the **Apply Changes** button to save your process.

3. Exercise #03: Create data model

First, we need to create a table to store the relevant user data during the execution of a single process instance. Upload the file **tuto_db_objects.sql** from the downloaded folder and upload it in the *SQL Scripts* section inside the *SQL Workshop* of your workspace and execute it afterwards to create the following objects:

- Table *tuto_expenses*
- View *tuto_p0001_vw*
- View *tuto_p0002_vw*
- View *tuto_p0003_vw*

4. Exercise #04: Create application

In this chapter we build the application for executing the defined business process.

Start by creating a new APEX application named *ExpenseApp*, keeping all the default settings.

We will deal with the automatically created page 1 later and start with the first additional page. For that we need a form page, that serves as entry point to the process by creating a new expense report. Create a new **Modal Dialog** page of the type **Form** called *Create Expense Report*. Choose the view *TUTO_P0002_VW* as **Data Source** and select *EXPE_ID* as **Primary Key Column**. All other settings can be left at their default.

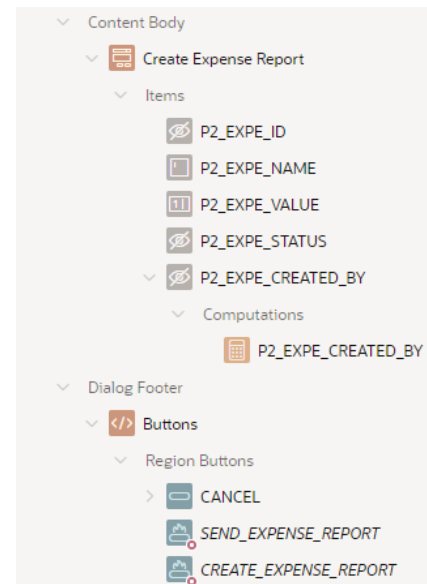
After creation, change the type of the items

- **P2_EXPE_STATUS**
- **P2_EXPE_CREATED_BY**

to *hidden* since they are not to be manually set by the end user.

Rename the labels of the displayed items

P2_EXPE_NAME and **P2_EXPE_VALUE** to *Report Name* and *Value*.



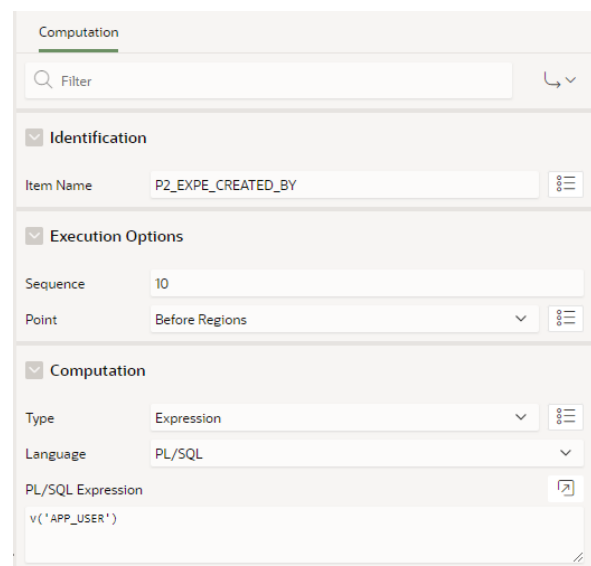
Next create a *Computation* on the item **P2_EXPE_CREATED_BY** item to retrieve the current user's name. Use the settings shown on the right.

In the buttons section of the dialog

- delete the button **DELETE**
- rename the button **SAVE** to *SUBMIT_EXPENSE_REPORT*
- rename the button **CREATE** to *CREATE_EXPENSE_REPORT*.

The code for processing the submitted forms is added later in chapter #05.

Save this page and **go to Page 1**.



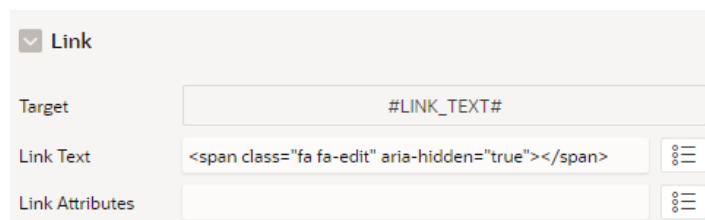
This page will contain a report showing all current process instances for a specific user and provide the links to the next page respective to the following user tasks.

On page 1, create a new region named *Expense Reports* of the type **Interactive Report**. Set its source to the previously created view *TUTO_P0001_VW*. Additionally, add the following **Where Clause**:

```
expe_created_by = :APP_USER or user_name = :APP_USER
```

This ensures, that the report shows only process instances started by the current user or inside the accessible lanes.

Change the type of the columns **EXPE_ID**, **PRCS_ID**, **USER_NAME** and **LINK_STYLE** to *Hidden Column* and the type of the column **LINK_TEXT** to *Link*. Enter the following values under the *Link* section (**target type** set to *URL*):



Link Text: ``

Rename all the displayed column's headings accordingly to their content (*Report Name*, *Value*, *Status*, *Created By* and *Link*).

Create a button named and labelled **Create** in this region and set its position to *Right of Interactive Report Search Bar*. Change its **behaviour** to *Redirect to Page in this Application* and set the recently created Page 2 as **target**.

Add a **Dynamic Action** which is executed when the *Dialog Closed* event occurs on the scope of the interactive report and use it to refresh the region *Expense Reports*.

Finally save your changes to this page.

In addition to the previous form to create and modify expense reports, we need another form for the approving and reviewing steps of the process.

Create another **Modal Dialog** page of the type **Form** called *Review Expense Report*. Again, take over the default settings and choose the view *TUTO_P0003_VW* as **Data Source** as well as the column *EXPE_ID* as **Primary Key Column**.

After creation, change the type of the items

- **P3_EXPE_NAME**
- **P3_EXPE_VALUE**
- **P3_EXPE_CREATED_BY**

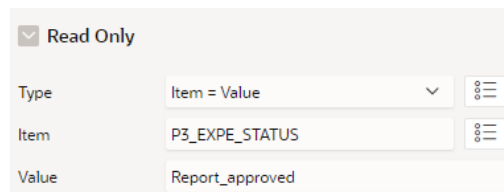
to *Display Only* and arrange them in a row.

The type of **P3_EXPE_STATUS** should be set to *Select List* to ensure process conform input values. Under the *List of Values* section insert the following input static values:

- Display: approve, Return: Report_approved
- Display: reject, Return: Report_rejected

The return values should match with the corresponding gateway routes in the modelled process. Additionally, enable the **Value Required** attribute under *Validation* on this item and set its **Template** under *Appearance* to *Required - Floating*.

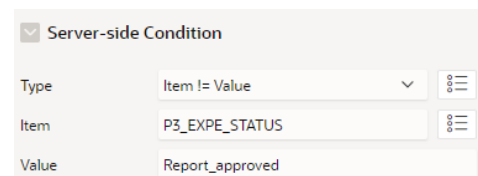
Also, add a condition for the **Read Only** attribute of the item using the following settings:



Type	Item	Value
Item = Value	P3_EXPE_STATUS	Report_approved

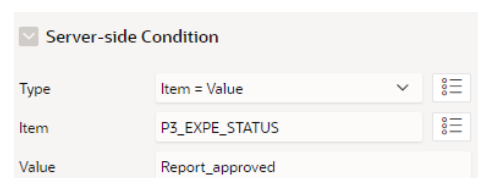
Like on the previous pages, rename the form field labels to *Report Name*, *Value*, *Created By* and *Status*.

In the buttons section of the dialog, delete the **DELETE** Button and rename the **SAVE** button to *REVIEW_EXPENSE*. Enter the **Server-side Condition** shown on the right.



Type	Item	Value
Item != Value	P3_EXPE_STATUS	Report_approved

Rename the **CREATE** button to *REVIEW_PAYMENT* and set the **Database Action** field under the *behaviour* section to *SQL UPDATE action*. Enter the displayed **Server-side Condition**.



Type	Item	Value
Item = Value	P3_EXPE_STATUS	Report_approved

Save the changes to this page and start with adding the processing elements for the created forms in the next section.

5. Exercise #05: Link application to flow

Prerequisites

To control the modelled flow and its single tasks from within your application, you need to be able to reference the current process ID, subflow ID and step key. Create three **Application Items** *PROCESS_ID*, *SUBFLOW_ID* and *STEP_KEY* under the *Shared Components*, so you can use the items on all your pages. Be sure to choose *Unrestricted* or *Checksum Required* for the **Security** setting of the items to allow them being set by the flow engine.

The functions, that shall be executed on the script and service tasks, are bundled inside a custom PL/SQL package. To install that package to your workspace, upload and execute the downloaded file *tuto_expense_pkg.sql* inside the **SQL Scripts** section.

Next, we need to copy the needed Flows plugins from the *Flows for APEX* app. We will use these plug-ins for processing the single tasks and displaying the process diagram inside our own application. In the *Shared Components* section of the *Expense App*, open the **Plug-ins** page and click the **Create** button. Select *As a Copy of an Existing Plugin* and click **Next**. Select the *Flows for APEX* application and click **Next** again. Now set the **Copy?** attribute to Yes for the following list of plugins:

- Flows for APEX - Manage Flow Instance
- Flows for APEX - Manage Flow Instance Step
- Flows for APEX - Manage Flow Instance Variables
- Flows for APEX – Viewer

The *Flows for APEX – Modeler* plugin does not have to be copied.

After that, go the **Component Settings** in the *Shared Components* and open the installed plugin *Flows for APEX - Manage Flow Instance*. Here you can enter a value for the component setting **Global Flow**. Enter the name of the modelled flow *ExpenseFlow* to set the needed reference to the model.

As Flows for APEX can be used with a minimum version of APEX 20.1, the plugin for the APEX 22.1 exclusive feature *Task Definitions* is not bundled inside the application. Instead, download the plugin *Flows for APEX – Return to Flows* file from

https://github.com/flowsforapex/apex-flowsforapex/blob/development/src/plugins/return-apex-approval-result/process_type_plugin_com_flows4apex_return_to_flows_process.sql

and import it into the *Expense App* as well.

Monitor process instance

To be able to monitor the current state of a specific flow instance, the viewer plug-in can be used to embed a diagram view inside the application.

Create a new blank **Modal Dialog** page named *Flow Monitor*. Add a new region to this page named equally and choose the installed **Flows for APEX - Viewer** plug-in as the region's type. Select the Flows for APEX specific view *FLOW_INSTANCE_DETAILS_VW* as the region's **source** and enter the following **Where Clause**, so the process ID application item is used to identify the current process instance:

```
prcs_id = :PROCESS_ID
```

Under the **attributes** tab of the plugin enter the values displayed on the right. Set the **template** under the *appearance* section of the region to *Blank with attributes* to show only the viewer canvas.

Finally enable the *Stretch to fit window* setting under the page's **Template Options**, so the diagram is shown in an appropriate size.

Save your changes to this page and **return to page 1**.

The 'Identification' tab is selected. It contains the following fields:

- Title:** Flow Monitor
- Type:** Flows for APEX - Viewer [Plug-In]
- Source:**
 - Location:** Local Database
 - Type:** Table / View
 - Table Owner:** Parsing Schema
 - Table Name:** FLOW_INSTANCE_DETAILS_VW
 - Include ROWID Column:** (toggle is off)
 - Where Clause:** prcs_id = :PROCESS_ID

The 'Settings' tab is selected. It contains the following fields:

- Diagram XML:** DGRM_CONTENT
- Add Highlighting:** (toggle is on)
- Current Nodes:** ALL_CURRENT
- Completed Nodes:** ALL_COMPLETED
- Error Nodes:** ALL_ERRORS
- Enable Call Activities:** (toggle is on)
- Diagram Identifier:** PRDG_ID
- Calling Diagram Identifier:** PRDG_PRDG_ID
- Calling Object ID:** CALLING_OBJT
- Breadcrumb Text:** BREADCRUMB
- Allow Drilldown On Diagram Level:** DRILLDOWN_ALLOWED
- Refresh On Load:** (toggle is on)
- Enabled Viewer Navigation:** (toggle is off)

Now we want to add a link to the previously created dialog page inside the report. For that we use the column **PRCS_ID** which we set to *Hidden Column* before. Change its type to *Link*, enter *Flow Monitor* as the column's **heading** and set the following options inside the **Target** dialog:

Link Builder - Target

☒ Target

Type: Page in this application

Page: 4

☒ Set Items

Name	Value
PROCESS_ID	#PRCS_ID#

☒ Clear / Reset

Clear Cache

Action: **None** Clear Regions Reset Regions Reset Pagination

Cancel Clear OK

Additionally, instead of displaying the process ID, add the following **Link Text**:
``

☒ Link

Target: Page 4

Link Text: ``

Link Attributes:

Cancel Clear OK

Finally save your changes to Page 1.

Processing forms

Switch back to the first created form **page 2** and open the processing tab. Under the *Processing* section you can find the automatically generated process for storing the expense report input data to the region source. Here we must add the flow-specific processes to control the single tasks inside our modelled flow.

Create a new process named *Flow – Create and Start* of the type **Flows for APEX – Manage Flow Instance**, the previously installed plug-in, and enter the settings show on the right. This provides a correct creation and initialization of the referenced flow.

For clearly identifiable process instances, you can use a combination of static text and Item Values as the **Instance Name**:

Expense Report -
&P2_EXPE_CREATED_BY. -
&P2_EXPE_NAME.

Change the **Server-side Condition** to *When Button Pressed* and choose the button *CREATE_EXPENSE_REPORT*. Ensure the correct execution order by putting the process element below the *Process Form* element.

The screenshot shows the configuration window for a process named 'Flow - Create and Start'. It is categorized under 'Identification' and 'Settings'. The 'Type' is set to 'Flows for APEX - Manage Flow Instance [Plug-I]'. Under 'Settings', the 'Action' is 'Create and Start'. The 'Flow Instance Name' is configured with the text 'Expense Report - &P2_EXPE_CREATED_BY. - &P2_EXPE_NAME.'. The 'Select Flow using' is set to 'Component Setting'. The 'Flow (Diagram) selection based on' is 'Name'. The 'Set Business Reference' is 'P2_EXPE_ID'. The 'Return Instance ID into' is 'PROCESS_ID'. The 'Set Process Variables?' is set to 'No Process Variables'.

Create a second process named *Flow - Complete Step* and put it below the previous one. This process is used for moving one step forward inside the flow and is needed to complete the *Correct Expense Report* User Task. Change its type to the previously installed **Flows for APEX – Manage Flow Instance Step** plugin and set the displayed values under the settings tab.

Like the first process, set the **Server-side Condition** to *When Button Pressed* and select the button *SUBMIT_EXPENSE_REPORT*.

The screenshot shows the configuration window for a process named 'Flow - Complete Step'. It is categorized under 'Identification' and 'Settings'. The 'Type' is set to 'Flows for APEX - Manage Flow Instance'. Under 'Settings', the 'Action' is 'Complete Step'. The 'Flow Instance info' is 'In Page Items'. The 'Process ID item' is 'PROCESS_ID'. The 'Subflow ID item' is 'SUBFLOW_ID'. The 'Step Key' is 'STEP_KEY'. The 'Set Gateway Routing?' and 'Auto-Branching?' options are both disabled (unchecked). The 'Return Flow Instance and Subflow ID' field is empty.

Remove the **Server-side Condition** from the *Close Dialog* process so it is always executed at the end.

Save this page and **switch to page 3**.

For this form page, additional processes are needed for completing a step. Add two processes of the **Flows for APEX – Manage Flow Instance Step** plugin type and use the following settings:

Identification

NameFlow - Complete Step with Gateway

TypeFlows for APEX - Manage Flow Instance Step [Plug-In]

Settings

ActionComplete Step

Flow Instance InfoIn Page Items

Process ID ItemPROCESS_ID

Subflow ID itemSUBFLOW_ID

Step KeySTEP_KEY

Set Gateway Routing?☒

Gateway IDReport_reviewed

Route ID&P3_EXPE_STATUS.

Auto-Branching?☐

Return Flow Instance and Subflow ID [deprecated]

Execution Options

Success Message

Error

Server-side Condition

When Button PressedREVIEW_EXPENSE

Type- Select -

Identification

NameFlow - Complete Step

TypeFlows for APEX - Manage Flow Instance Step [Plug-In]

Settings

ActionComplete Step

Flow Instance InfoIn Page Items

Process ID ItemPROCESS_ID

Subflow ID itemSUBFLOW_ID

Step KeySTEP_KEY

Set Gateway Routing?☐

Auto-Branching?☐

Return Flow Instance and Subflow ID [deprecated]

Execution Options

Success Message

Error

Server-side Condition

When Button PressedREVIEW_PAYMENT

Type- Select -

On the left processing, be sure to enter the correct gateway id *Report_reviewed* and use the page item substitution *P3_EXPE_STATUS*. so the correct gateway path is taken automatically after completing the task.

Again, remove the **Server-side Condition** from the *Dialog Closed* process and save your changes to the page.

bpmn modeller settings

Inside the bpmn modeller you can use additional information and settings to call APEX pages or run scripts inside your application. Open your modelled *ExpenseFlow* inside the **Flow Management Section** of the Flows for APEX application by using the row menu option **Show Details** in the overview. Click the **Modify Diagram** button on the next page to open the modeler.

User Tasks

User Tasks are designed to open a specific APEX page when reached inside the process flow by providing a generated link to that page. Click the first user task *Review expense* and open the **APEX Page** tab in the settings panel on the right.

Here you can specify the Application where your Flow Diagram is used as well as the page to execute the User Task. You can use the APEX metadata here, to directly select your created application and page from the select lists. Alternatively, you can manually type in the needed information. In that case, be sure to use the correct application ID of the *ExpenseApp* here. Alternatively, you can leave the field blank to use the currently opened application. *Note: On apex.oracle.com the usage of the alias can cause errors when querying the generated link to the APEX page. We recommend using the meta data directly here or manually put in the application ID (or leaving it empty).*

By setting the **Page Item** and **Item Values**, references to the current process ID, subflow ID and step key are granted. You can fill in the default key-value pairs by clicking **Generate Default** or manually add needed items. The ID to the explicit expense entry will be set as the business reference variable during the creation of the process instance inside the plugin and can be accessed via the substitution string `&F4A$BUSINESS_REF.`

Enter the following values to create a connection to page 3:

- Input: Use APEX meta data
- Application: ExpenseApp
- Page: Review Expense Report
- Page Items:

Item Name	Item Value
PROCESS_ID	&F4A\$PROCESS_ID.
SUBFLOW_ID	&F4A\$SUBFLOW_ID.
STEP_KEY	&F4A\$STEP_KEY.
P3_EXPE_ID	&F4A\$BUSINESS_REF.

After manually adding a new item, you can use the quick link to populate the item with the right substitution syntax for built-in variable (PROCESS_ID, SUBFLOW_ID, STEP_KEY and BUSINESS_REF).

Review_expense

< General **APEX Page** Variable Express >

Call APEX Page

Input

Use APEX meta data

Application

ExpenseApp

Page

Review Expense Report

Page Items

PROCESS_ID : &F4A\$PROCESS_ID.
SUBFLOW_ID : &F4A\$SUBFLOW_ID.
STEP_KEY : &F4A\$STEP_KEY.
P3_EXPE_ID : &F4A\$BUSINESS_REF.

Generate defaults

Item Name

P3_EXPE_ID

Item Value

&F4A\$BUSINESS_REF.

process_id subflow_id step_key
business_ref

Request

Request Value for Page Call

Clear Cache

Clear Cache Value for Page Call

Item Name

P2_EXPE_ID

Item Value

&F4A\$BUSINESS_REF.

process_id subflow_id step_key
business_ref

Similarly change the settings for all existing User Tasks:

Review for payment

- Input: Use APEX meta data
- Application: ExpenseApp
- Page: Review Expense Report
- Page Items:

Item Name	Item Value
PROCESS_ID	&F4A\$PROCESS_ID.
SUBFLOW_ID	&F4A\$SUBFLOW_ID.
STEP_KEY	&F4A\$STEP_KEY.
P3_EXPE_ID	&F4A\$BUSINESS_REF.

Correct expense report

- Input: Use APEX meta data
- Application: ExpenseApp
- Page: Create Expense Report
- Page Items:

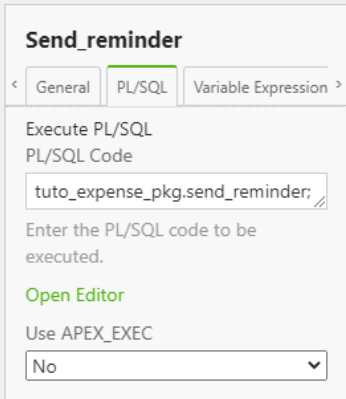
Item Name	Item Value
PROCESS_ID	&F4A\$PROCESS_ID.
SUBFLOW_ID	&F4A\$SUBFLOW_ID.
STEP_KEY	&F4A\$STEP_KEY.
P2_EXPE_ID	&F4A\$BUSINESS_REF.

Script Tasks / Service Tasks

On Script and Service Tasks you can trigger the execution of PL/SQL code by invoking functions and procedures of custom packages. The settings can be entered under the **PL/SQL** tab after clicking a task in the modeller.

Use the following procedure calls as **PL/SQL code** for the Script and Services Tasks in the model:

- Send reminder: *tuto_expense_pkg.send_reminder;*
- Advise employee of rejection:
tuto_expense_pkg.advise_employee;
- Make payment: *tuto_expense_pkg.make_payment;*
- Send payment advice: *tuto_expense_pkg.finish_expense;*



Send_reminder

< General **PL/SQL** Variable Expression >

Execute PL/SQL

PL/SQL Code

tuto_expense_pkg.send_reminder;

Enter the PL/SQL code to be executed.

[Open Editor](#)

Use APEX_EXEC

No

Finally save your changes to the flow diagram by clicking the **Apply Changes** button.

6. Exercise #06: Define user roles

The mapping of the three lanes inside the modelled flow can be realised by using the APEX user role mechanism. By that you can ensure that each user gets access only to the current steps inside its permitted area.

Open the **Administration** page inside your workspace and click **Manage Users and Groups**. Create three new users as end users using an accessible mail address of yours and free to choose names and passwords. To maintain consistency when assigning the roles, you can name them accordingly to the lanes inside the modelled flow or by using shortcuts (e.g., EMP, FLM, FEP).

Return to the *ExpenseApp* and go to **Application Access Control** in the *Shared Components*. Create three new **Roles**, named identically to the lanes in the model:

- Employee (Static ID: EMPLOYEE)
- First Line Manager (Static ID: FIRST_LINE_MANAGER)
- Finance Expense Processing (FINANCE_EXPENSE_PROCESSING)

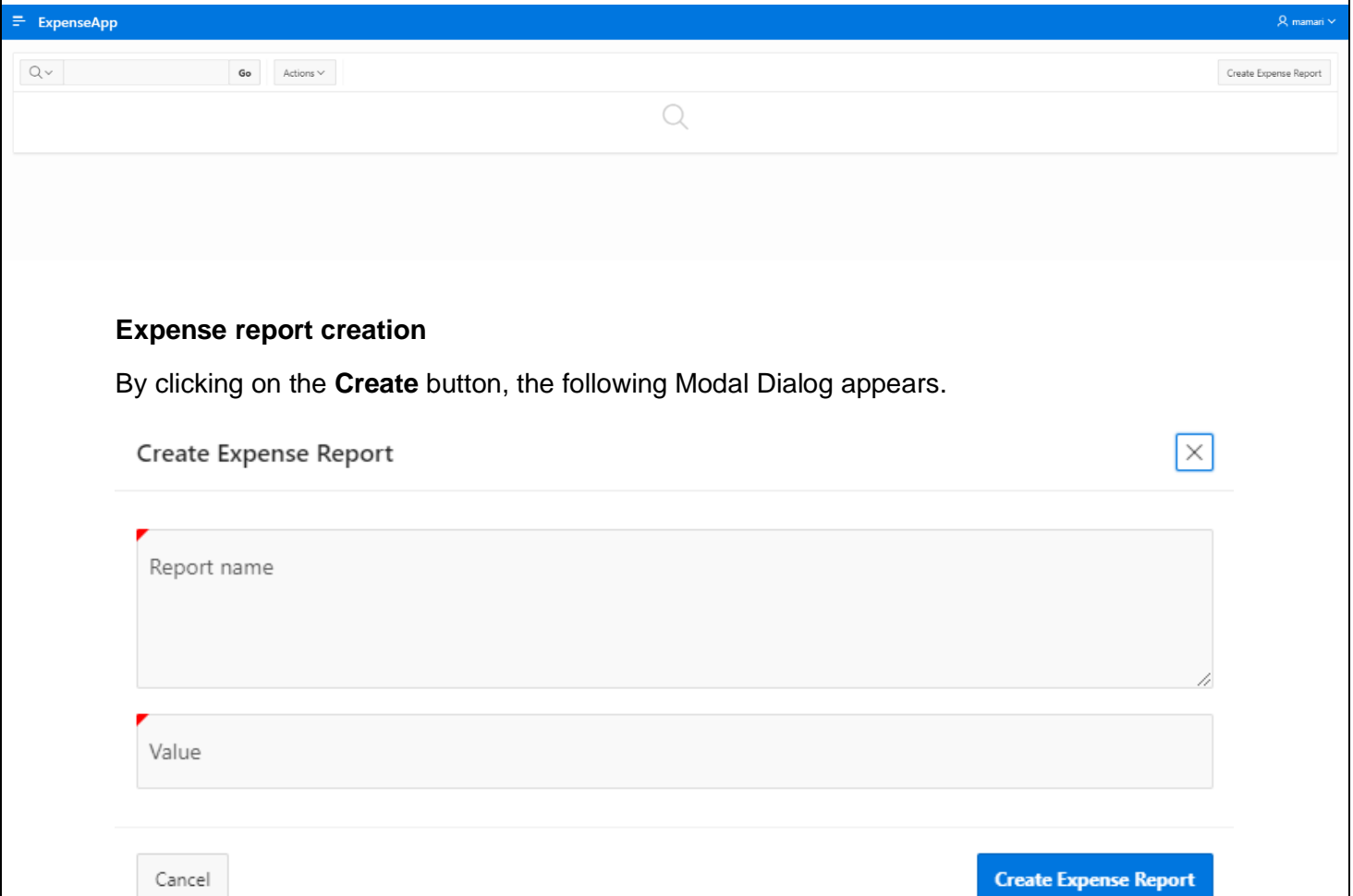
Then add the user role assignments by adding each role to one of the previously created users.

Important note: *By using this way of role management, in general all users sharing a lane can access all current process instances in that lane. This is due to the logic behind lanes in BPMN processes. In this case, each employee can see all created expense report entries, while the users in first line manager and finance expense processing lane can see all existing entries. This is because the view on page 1 already uses filtering on the EXPE_CREATED_BY column. On the other side, this allows us to combine the employee's and manager's tasks to one single form page since there are no accessibility issues between them. To enable further authentication, you can use the audit column in combination with a strictly implemented hierarchy between employees and managers with proper page and region visibilities as well as the APEX authorization schemes. For bigger and more complex business applications, Flows for APEX allows the reservation of single process tasks to show other users, that an instance is already being processed internally. Thus, concurrent operations on a single process instance can be prevented. Notice that this functionality as well as further authorization techniques are not included as part of this tutorial for the sake of simplicity.*

7. Exercise #07: Testing

In this chapter we want to test our application in combination with the modelled flow.

Start by logging in into the **ExpenseApp** with a user from the *Employee* group.



The image shows the ExpenseApp interface. At the top is a blue header bar with the text "ExpenseApp" on the left and a user profile icon labeled "mamani" on the right. Below the header is a white navigation bar containing a search icon, a "Go" button, an "Actions" dropdown menu, and a "Create Expense Report" button. The main content area is white and contains the "Expense report creation" section. This section has a title "Expense report creation" and a description: "By clicking on the **Create** button, the following Modal Dialog appears." Below this description is a modal dialog titled "Create Expense Report" with a close button (X) in the top right corner. The dialog contains two input fields: "Report name" and "Value". At the bottom of the dialog are two buttons: "Cancel" and "Create Expense Report".

Expense report creation

By clicking on the **Create** button, the following Modal Dialog appears.

Create Expense Report

Report name

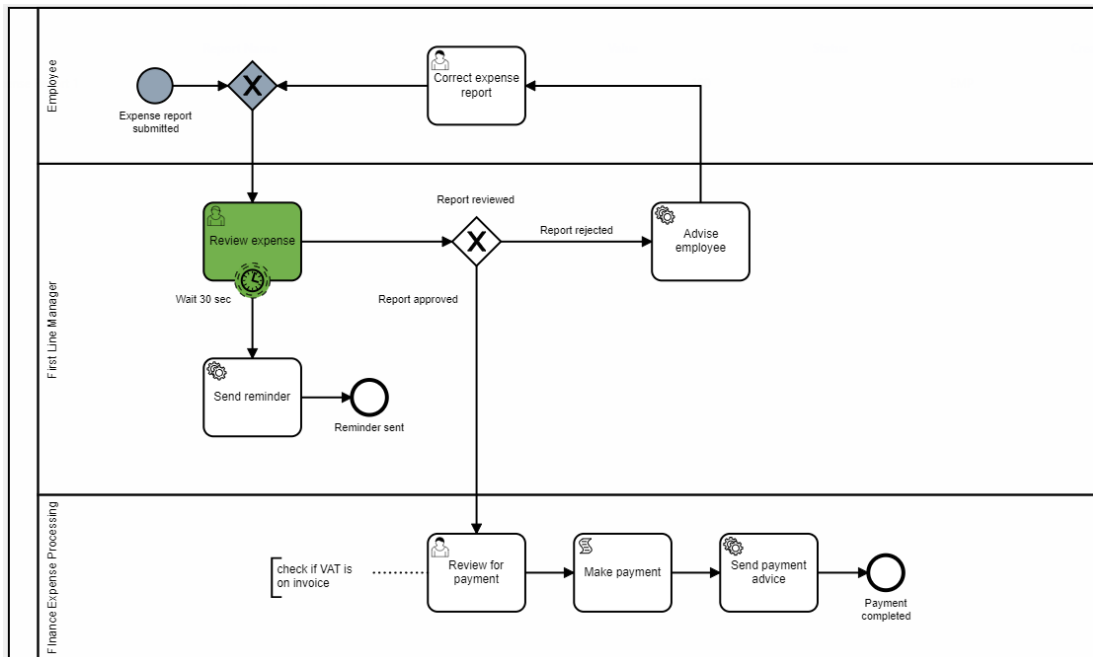
Value

Cancel Create Expense Report

Fill the required input fields and click the button **Create Expense Report** to submit the data and create a new entry.

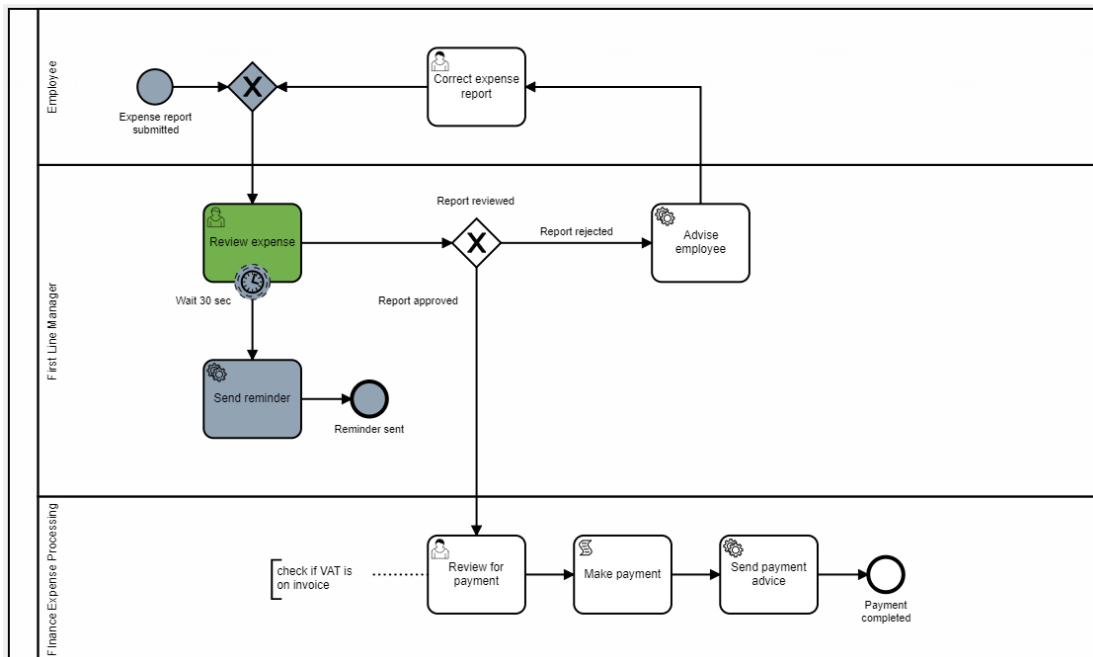
The diagram of the process instance can be displayed by clicking the **eye** icon in the corresponding row.

The diagram for a newly created process instance will look as follows:



As shown in the viewer, the timer attached to the user task has been started as well.

After the timer's end, you will notice that all *First Line Manager* group users received a reminder email.



You are required to log in as a *First Line Manager* group user now to proceed.

Expense report approval

Proceed the flow by clicking the link column again to review the expense report.

Review Expense Report

Report Name

My Expense Report

Value

100

Created By

EMP

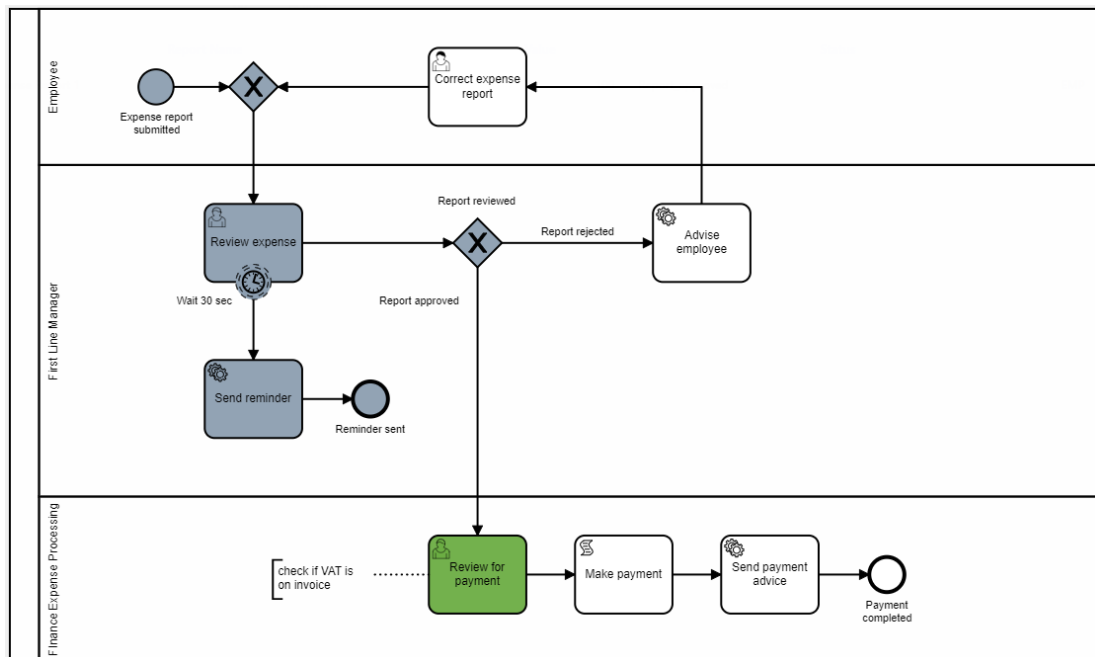
Status

approve

Cancel

Review Expense

After approving the expense report, the current activity on the Flow Monitor moved into the *Finance Expense Processing Line*.



Expense report payment review

You need to log in with a user from *Finance Expense Processing* group and review the payment as shown below.

Review Expense Report

Report Name

My Expense Report

Value

100

Created By

EMP

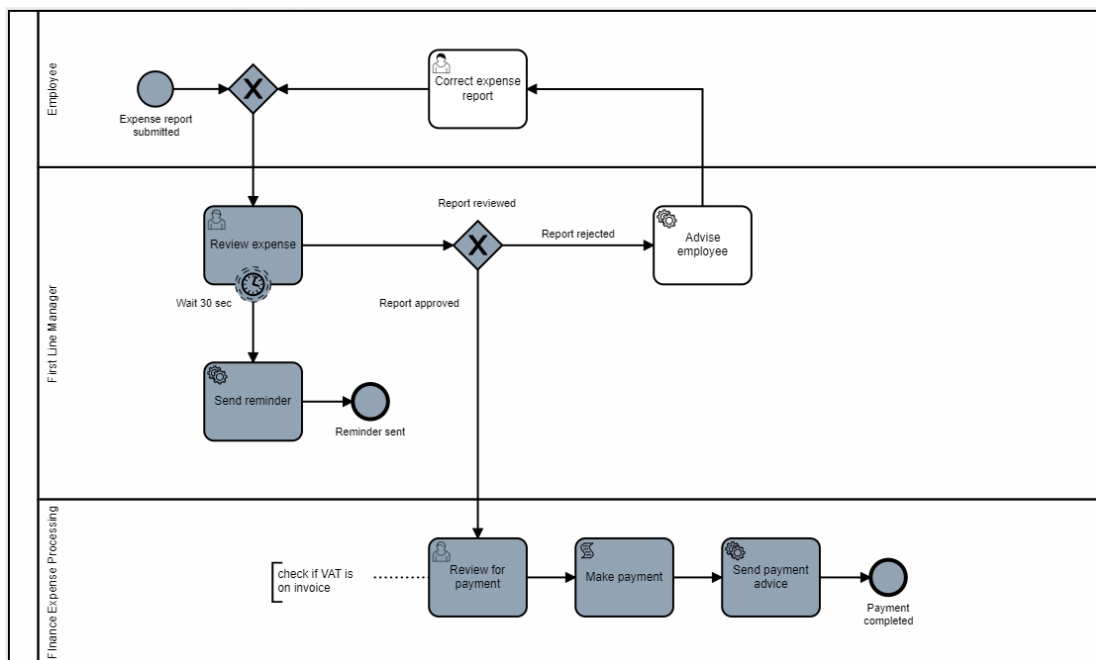
Status

approve

Cancel

Review Payment

That will end the flow and you will notice that an email was sent to confirm the payment.



Expense report rejection

You can repeat the steps until the review of the expense report and choose to reject the report this time.

Review Expense Report

Report Name

My Expense Report 2

Value

200

Created By

EMP

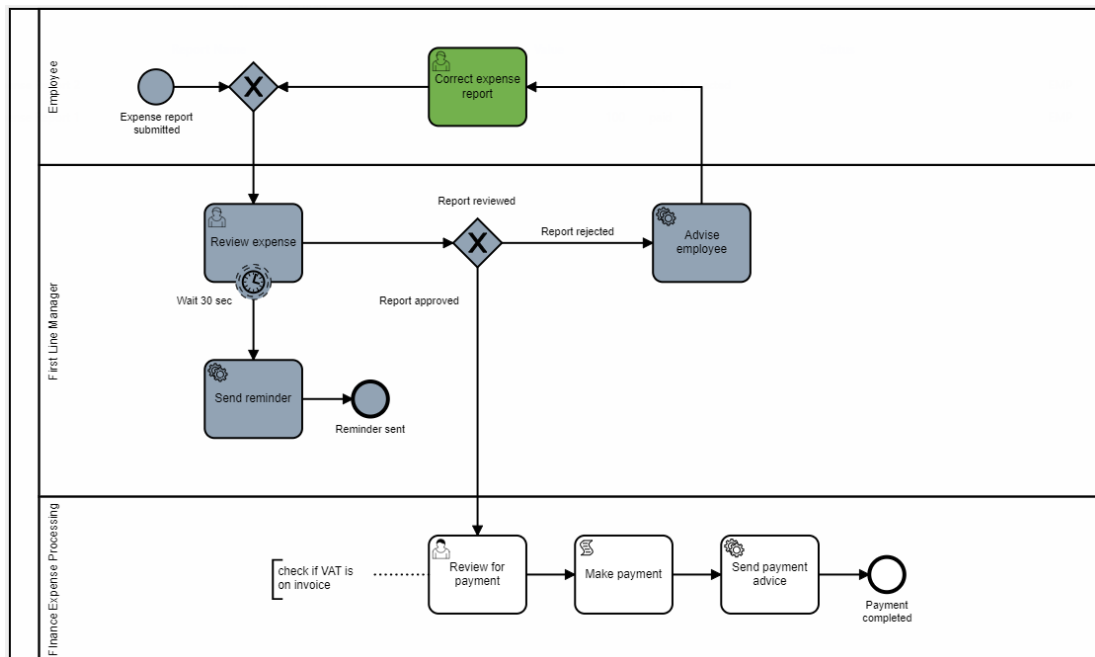
Status

reject

Cancel

Review Expense

This will bring you the into the **Correct expense report** activity after receiving rejection email.



You can correct your report and send it again in to be reviewed by a user from the *First Line Manager* group.

8. Exercise #08: Versioning

In Flows for APEX each process instance is based on a diagram name and a version. This allows you to have multiple versions of your flow while ensuring the runnability of the single instances. Additionally, the status attribute is used to mark the current version of your diagram and if changes to the diagram can be expected.

Each newly created diagram will start with the status *draft*. This status is typically used during the modelling phase, where you can edit and test your diagram at any time without changing the version. Once your flow is ready for production, you can set its status to *released*, which makes it immutable. For further changes you would have to create a new version of the diagram, starting with *draft* status again.

Open the *Flows for APEX* application and go to the **attributes** panel of your modelled flow.

Flow Management \ ExpenseFlow - Version 0 Apply Changes

Attributes		Process instances per status	
Category Tutorial		0 Created	0 Running
Name ExpenseFlow		0 Completed	
Version 0	New Version	0 Terminated	0 Error
Status draft	Release		

Click the button **Release** to change the status of this diagram to *released*. The **Modify Diagram** button is now hidden to prevent changed to your diagram.

Open the *Expense App* and start a new process instance by creating an expense report. If you do not specify a diagram version in the Component Settings, always the released version of the diagram will be used. Otherwise, if no *draft version 0* exists, errors might occur.

Now we want to create a new version of the diagram. Go back the *Flow Modeller* and click the button **New Version**. Enter *1* as the **version** identifier and click **Add Version** to close the dialog. A new version of the diagram has been created. Click the **Modify Diagram** button and make some minor but visible change to the diagram, for example change the timer text and definition from *30 seconds* to *60 seconds*.

To release this new version of the flow, the old version must be set to *deprecated* first. This can be done by clicking the **button** next to the status field on the attributes panel of the version 0 diagram. Now you can change the status of the new diagram to *released*.

Inside the *Expense App* create another process instance. By comparing the corresponding diagram windows using the viewer plugin you can see that the first entry still uses the deprecated diagram, while the second entry is based on the new version instead.

After all existing instances of a deprecated diagram have finished, you can set the diagram status to *archived* to indicate that it is not used for processing anymore.

9. Exercise #09: Process Variables

Process variables provide a way to store process relevant information throughout the whole execution of a single instance. They are stored persistently in the database and hence (in contrast to APEX items) can be used without an APEX session.

In this demo, process variables for a flow instance already have been used to store the business reference and the path to choose for the gateway, depending on the user's input. This works automatically by using the pre-configured attributes in the process plugins.

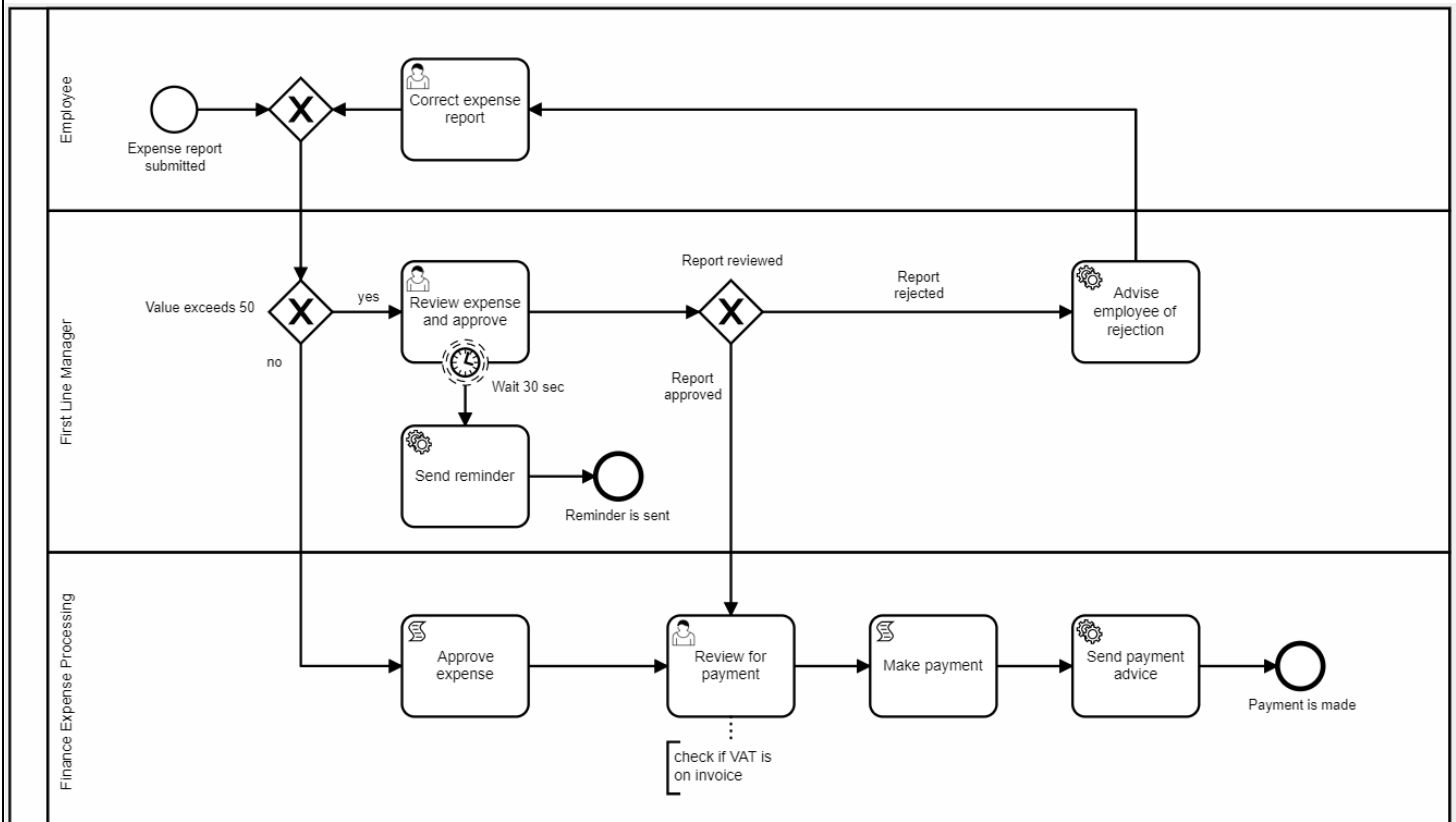
Additionally, you can define and set process variables in the flow modeler.

In this chapter we want to modify the approval process in a way, that the First Line Manager only has to approve expenses, when the value exceeds a certain limit.

First, open the modelled flow in the **Flow Modeler**. Extend the process flow as shown below by adding an additional gateway. The script task in the *no* path is used to update the expense and set the status to approved, in case no manual approval by a first line manager is needed. Be sure to set the IDs of the gateway, task, and the routes as before by following the described rule set:

- underscore notation (Approve expense -> Value_exceeds_50)
- avoid implementation details (Value exceeds 50 -> Value_exceeds_limit)

Also, keep in mind that all IDs must be unique and therefore using *yes* and *no* for the sequence flow could cause issues. Instead, you can use parts of the prior gateway in the flow identifiers, e.g. *Limit_exceeded* / *Limit_not_exceeded* or *Exceeded_yes* / *Exceeded_no*



Gateway routing expressions

With Flows for APEX you can control gateway paths by using routing expressions. On every outgoing gateway path, you can specify a condition when this path should be taken. If one (for exclusive gateways) or multiple (for inclusive gateways) path conditions evaluate to *true*, the gateway is completed and the workflow proceeds.

In comparison to older versions of Flows for APEX, this makes it easier to model gateway logic as you don't have to specify an explicit process variable containing the gateway route. You can still use this approach in your models as the process variable will always have priority over the path expressions. Therefore, you need to set a variable named `<gateway_id>:<route_id>` by using the bpmn ids to set the route(s) before the gateway gets executed.

In this tutorial we will use path expressions instead. We will store the expense value in a process variable and read out this variable in the expressions to determine, which path should be taken.

In the still opened **Flow Modeler**, select the *yes* route going out of the newly modelled gateway to open the properties panel. Under the *Condition* section you can see that the **Sequence** has been set automatically. You can change this if needed to control the evaluation order of all your paths. Select *Expression* as the **Condition Type** and enter the following *expression* in the **Condition** field:

```
:F4A$EXPE_VALUE > 50
```

In gateway routing expression you can use bind variable syntax, so you don't need to use the substitution syntax here.

In the same way set the expression on the *no* path to:

```
:F4A$EXPE_VALUE <= 50
```

You can also leave the condition for this path blank, as soon as the *yes* path has a lower sequence number und thus gets evaluated first.

Now select the script task and enter the following call in the **PL/SQL code** section:

```
tuto_expense_pkg.approve_expense;
```

This procedure will update the status of the current expense to *approved*.

Save the changes to the model, go back to the App Builder and **open page 2** of the *ExpenseApp*.

The process variable *EXPE_STATUS*, that is used inside the expressions, has to be set at two points: after creating a new process instance and after completing the *Correct Expense Report* User Task.

The first scenario can be solved in the already created processing for *Flow – Create and Start*. In the property's settings section, set the **Set Process Variables** option to *Using JSON* and paste the following JSON array into the appearing text box:

```
[
  {
    "name": "EXPE_VALUE",
    "type": "number",
    "value": "&P2_EXPE_VALUE."
  }
]
```

The process variable will now be set directly in the course of creating a new entry with a reference to the started process instance.

For the second scenario we need to add a new process on that page. Create a new process called *Flow – Set Variable* and select the plugin **Flows for APEX – Manage Flow Instance Variables** as type. Enter the shown settings and place it right above the *Flow – Complete Step* process, so it gets executed before that.

Set the **Server-Side Condition** to *When Button Pressed* and select the button *SUBMIT_EXPENSE_REPORT*, so it gets only executed when an employee corrects an expense report.

The updated value of an expense report will now be copied to the (at that point already existing) process variable after submitting.

Identification	
Name	Flow - Set Variable
Type	Flows for APEX - Manage Flow Instance Variable
Settings	
Action	Set
Flow Instance info	In Page Items
Process ID Item	PROCESS_ID
Manage Variable(s) using	APEX item(s)
Process Variable(s) Name(s)	EXPE_VALUE
APEX item(s)	P2_EXPE_VALUE
Return Flow Instance ID into	

Save your changes to the page and go back to the application.

If you now create an expense report with a value below 50, the review step of the First Line Manager will be skipped, and the process continues at *Review for payment*. The same occurs after a manager has rejected and the employee corrects the value to something below 50.

10. Exercise #10: Error handling

Starting with Flows for APEX 21.1, the engine is able to detect errors while processing a modelled flow and display the error information to the user. Using the **Flow Monitor** of the Flows for APEX app, an admin can then analyse, where the error occurred and initiate the needed error handling routines or fix the error directly by using the Flow Modeler. After that, a single step can be restarted so that the Flow can continue properly.

In this exercise we will manually enter an incorrect PL/SQL code on a Script Task and fix it afterwards, to test out the logging and restart functionality.

Open the **Flow Modeler** and edit your modelled flow. Select the *Make payment* script task and change the **PL/SQL code** to something, that will not execute, e.g., by adding a trailing **s** to the function call:

```
tuto_expense_pkg.make_payments;
```

Save your changes and open the *Expense App*. Create a new expense report and move forward in the process until the task *Review for Payment* has been completed. Now open the Flows for APEX Engine App and go to the **Flow Monitor** page (third entry in the navigation menu).

The *Flow Instances* report contains an overview of all the current instances. You will see that the **Status** column of your current instance is marked as *error* and has a red background. Click on the **Flow Name** of this instance to open the diagram in the viewer. You can see that the *Make payment* task is marked red as well.

▼

☑

🟢

Status = 'error'

✕

☑

☰

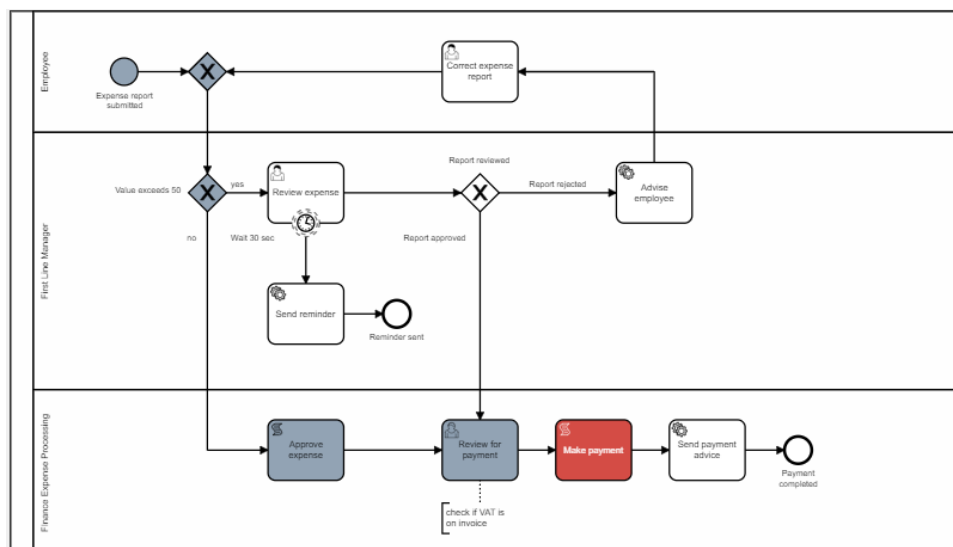
Model Category

✕

Model Category:

☐	☰	Quick Action	Name	Business Reference	Model Name	Model Version	Status	Creation Date	Last Update
☐	☰	🔍 Details	Expense Report - EMP - My Expense Report 3	3	ExpenseFlow	0	❗ error	26.09.2022 16:30:48	26.09.2022 16:34:07

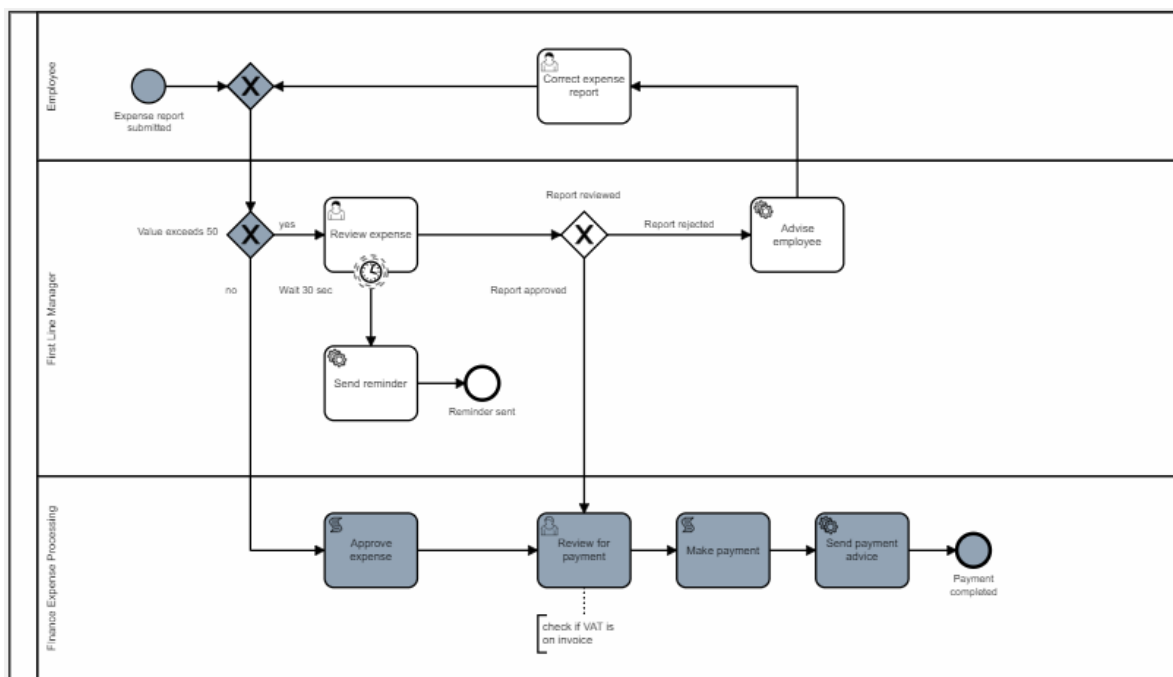
1 - 1



If you click on the task, you can see detailed information like the PL/SQL code that gets executed on this Script Task. Close this dialog and click on the **Details** link in the *Quick Action* column for this instance. On this page you can see more information about the instance as well as the current existing subflows and process variables. Click on the **Show History** button in the top right corner. This will open the event log for the current instance, including all the logged information on instance, subflow and step level. In the *error stack* column, you can now see, what error has happened in the process.

Expense Report - EMP - My Expense Report 3				
Instance Events Current Steps Completed Steps Variable History				
<input type="text"/> <input type="button" value="Go"/> <input type="button" value="Actions"/>				
User	Timestamp	Event	Comment	Error Stack
FEP	26.09.2022 16:34:07	error	Process 3: Task Make_payment failed due to PL/SQL error - see event log.	ORA-06510: PL/SQL: unhandled user-defined exception ORA-06512: at "WKSP_DAMTHORFLOWS.FLOW_PLSQL_RUNNER_PKG", line 153 ORA-06550: line 2, column 18: PLS-00302: component 'MAKE_PAYMENTS' must be declared ORA-06550: line 2, column 1: PL/SQL: Statement ignored ORA-06512: at "WKSP_DAMTHORFLOWS.FLOW_PLSQL_RUNNER_PKG", line 59 ORA-06512: at "WKSP_DAMTHORFLOWS.FLOW_PLSQL_RUNNER_PKG", line 134
EMP	26.09.2022 16:30:48	started		
EMP	26.09.2022 16:30:48	created		

Close this dialog and fix the error in the **Flow Modeler** by changing the PL/SQL statement back to the correct spelling. Save the diagram, go back to the **Flow Monitor**, and open the *Details* page of the instance again. Now we want to **restart** the *Review Payment* task so the process can continue. This can be done by clicking the *Quick Action* in the corresponding row of the *Subflow* report on the left. Enter an optional comment on the restart action and confirm the dialog. You can now see that the task has been successfully executed and the process could be finished.



11. Exercise #11: Email templates

Right now, all the service tasks in the model which are responsible for sending notification or reminder emails are using custom pl/sql calls for that. Inside the called procedures, we use `apex_mail.send` in combination with queried information about the current process instance.

Alternatively, you can use prior defined APEX email templates and specify all the needed settings for sending out emails declaratively inside your process model.

We now want to change the behaviour of the *Send Reminder* Service Task to use a template. To do this, we first have to create this template inside the ExpenseApp to be able to select it from the metadata afterwards in the modeler.

Go to the *Shared Components* and create a new **Email template** called *REMINDER*. Enter *Reminder Review Expense Report* as **Email Subject** and add the following html block in the **body** section:

```
<h1>Reminder</h1>
<p>Please review expense report created by #EXPE_CREATED_BY#.<p>
```

Besides static text we use a placeholder here which we will set to the value of a process variable later.

Click **Create Email Template** to save the template, go back to the **Flow Modeler** and open the *ExpenseFlow*.

Click on the *Send Reminder* Service Task to open the properties panel. In the *General* tab you can switch its **type** from *Execute PL/SQL* to *Send Mail*. Open the new appeared tab *Mail* in the properties panel. For the *sender* enter your own private email address. The recipient will be obtained by a process variable, so provide a substitution string here:

&F4A\$EXPE_CREATED_BY_MAIL.

If you scroll down, you can see the content related attributes. Set **Use Template** to Yes, so the select lists for choosing the template appear. Select *ExpenseApp* as application and the created *Reminder* template here. Below, you can specify any process-specific data that shall be passed into the email text. You can click **Load JSON** to automatically retrieve any defined placeholders in the template and automatically set up the JSON structure for you. Click **Open Editor** afterwards to provide the needed value for the *EXPE_CREATED_BY* attribute, which will again be the substitution string for a process variable:

"&F4A\$EXPE_CREATED_BY."

Pay attention to use double quotes this time to create a valid JSON syntax. Close the editor by clicking the green **Save** button and finally click **Apply Changes** to save your changes to the model.

Now we have to set the new process variables which are needed for sending out the mail. Go back to the App Builder and open **page 2**.

Content

Use Template

Yes

Input

Use APEX meta data

Application

ExpenseApp

Template

REMINDER

Placeholder

```
{
  "EXPE_CREATED_BY":
    "&F4A$EXPE_CREATED_BY."
}
```

Provide values for email template

Open Editor Load JSON

Attachment

SQL query to get attachment

Open Editor

First, we have to create a new page item to store the current user's email address. Create a new item *P2_EXPE_CREATED_BY_MAIL* of **type** *hidden* in the form region, add a *Computation* on it and add the following query (return single value):

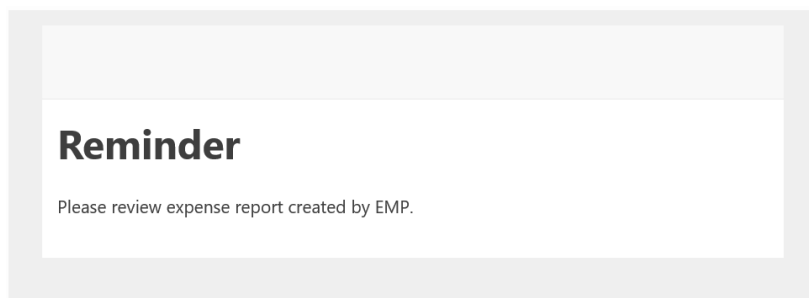
```
select email from APEX_WORKSPACE_APEX_USERS where user_name = v('APP_USER')
```

Open the processing section of the page where we added the process variable parts before. This time, we only have to make changes to the *Flow – Create and Start* process because we don't expect any of the information (name and email of the creator) to change during the process. Open the Code Editor on the *JSON* attribute on that process (where we added the *EXPE_VALUE* part earlier) and override it with the following JSON array:

```
[
  {
    "name": "EXPE_VALUE",
    "type": "number",
    "value": "&P2_EXPE_VALUE."
  },
  {
    "name": "EXPE_CREATED_BY",
    "type": "varchar2",
    "value": "&P2_EXPE_CREATED_BY."
  },
  {
    "name": "EXPE_CREATED_BY_MAIL",
    "type": "varchar2",
    "value": "&P2_EXPE_CREATED_BY_MAIL."
  }
]
```

We now store all the needed information for the email inside process variables on process creation and hence can use them inside the model.

Save your changed to the page and test it out by creating a new report and waiting for the timer to fire. You will now receive an email looking as depicted below, as specified in the template.



12. Exercise #12: APEX Approval Task

APEX Approval Tasks are a simple declarative way to manage basic approval processes inside your APEX application without a big overhead. You can create task definitions for several use-cases and show users their currently active tasks using an inbuilt overview page.

Flows for APEX 22.2 brings a support to include this functionality into your process models by adding approval task as a subtype for default user tasks. You create prior defined tasks from within your workflow and, after the work on the task has done, return to the running process.

You can use the inbuilt APEX functionalities to generate simple dialogs for approving and rejecting a task, which frees from the need to manually create form pages for all user tasks.

In this example we now want to use an approval task for *Review for payment*. The user should get the opportunity to approve or reject the payment, like the first line manager, but without the need of a separate database column for the status.

First, we need to create the task definition. Open the *Shared Components* of the *Expense App* and then the **Task Definitions** under the *Workflow and Automations* section. Click the **Create** button to create a new task definition. Enter *Review Payment* as **name** and *Review Payment &APEX\$TASK_PK*. As **subject** and leave the other attributes unchanged. For testing purposes, we add the task primary key, which will be the expense business reference, to the subject here, so we can see how the tasks and the business data are connected. Click **Create** to close the dialog and open the *Task Definition*.

Under *Settings* you can declare, what page should be opened when a user clicks on an active task. Here you can use the APEX inbuilt features to generate a page for you, that opens as a slide-in window and contains all the detailed information of the current task. Click the button **Create Task Details Page** next to the *Task Details Page Number* field to create the page and fill in the url. Go back to the *Task Definition* after that (you can check the *Return to page* option in the right bar on the page, so the page remains open after saving).

In the *participants* section you have to specify, which users are allowed to be the potential owner for this task, i.e., which users this task can be assigned to. In this simple example, we can hardcode the name of the user with the *Finance Expense Processing* role here (e.g. *FEP*, depending on the user's name defined in chapter #07).

Under *parameters* we need to add the process id from the running instance so we can return to that instance after the task has been completed. Enter *PROCESS_ID* as the **Static ID** and keep the suggested values for all the other options. Click **Apply Changes** before continuing.

The *actions* section contains all the processing parts that will be executed when a task changes its status. In this case we want to return the outcome of the task back to Flows for APEX when a user completes a task. Click the button **Add Action** to define a new action.

The first action shall be executed if the task is completed with the outcome *approved*. Enter *Return to Flows for APEX (approved)* as name and select the imported plugin **Flows for APEX – Return to Flows for APEX** as the type. Set the **On Event** attribute to *Complete* and select *Approved* as **Outcome**. Optionally you can provide a success message here, that will be shown in the top right corner inside the application, e.g., *Payment has been approved*.

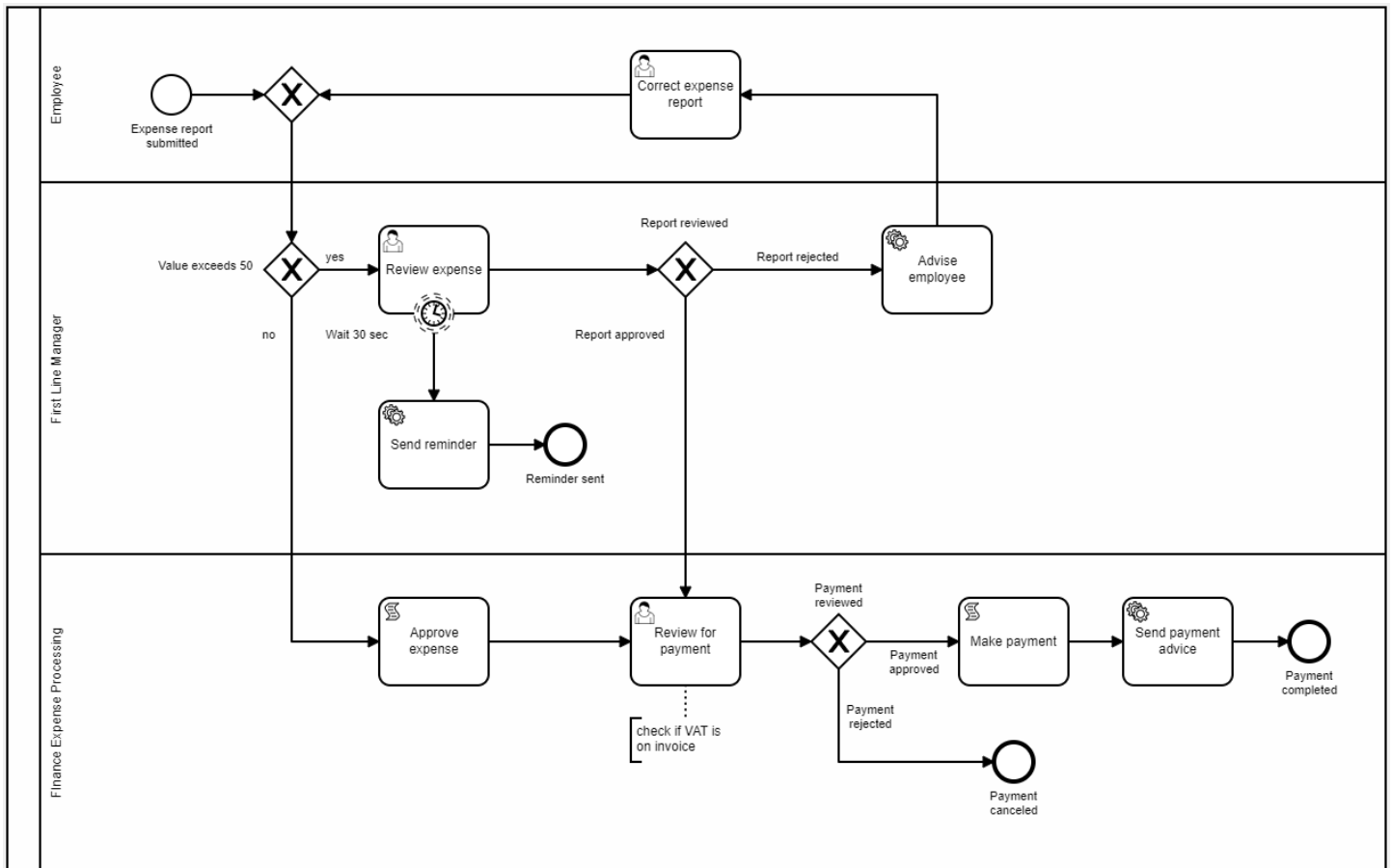
Click **Create** to save this action and return to the *Task Definition*. In the same way, create an action named *Return to Flows for APEX (rejected)* for the case, when the task is rejected.

Finally, you can click **Apply Changes** to save the task definition and open the **Flow Modeller** again.

Click the task **Review for payment** and change the **User Task Type** to *Apex Approval* in the properties panel. The task settings can be done in the now shown tab *APEX Approval*. Select the *ExpenseApp* and your **task definition** from the list and use the **quick picks** to automatically set the business reference and the parameters.

In the **Result Variable** field, we specify the name of the process variable where we want to store the return value from the APEX approval task. In this case we name it *PAYMENT_REVIEW*.

Next, we will add a gateway that controls the flow based on the task outcome. Add a new exclusive gateway as seen below and use the prior syntax for names and IDs. Now we use path expressions again to specify, which gateway path should be executed. In the condition section of the paths enter the following values:



- Path: payment approved
Condition Type: Expression
Condition: :F4A\$PAYMENT_REVIEW = 'APPROVED'
- Path: payment rejected
Condition Type: Expression
Condition: :F4A\$PAYMENT_REVIEW = 'REJECTED'

Now click **Apply Changes** and got back to the *Expense App App Builder*. As a last step we want to create an overview page to manage all the open tasks that come from the APEX Approval Tasks. Again, this can be done by using the default APEX templates. In the *App Builder*, click **Create Page** and select *Unified Task List* as the **Component**. Name it *Unified Task List* and keep the default setting for the *report context*.

Now log in to your application as an *Employee* to create a new test case. Proceed the flow until you logged in with the *Finance Expense Processing* user. You will see that this user doesn't have a new entry in the report on page 1 anymore. Instead, if you open the **Unified Task List** from the navigation menu, you can see, that there is an open task listed here.

Click the task header to open the detail page. At the bottom you can **Approve** or **Reject** this task. After you chose either of that, the task completes and disappears from the task list. In the **Flow Monitor** you can see that the user task completed as well, and the process continued to the end.

13. Exercise #13: Call Activities

Call activities offer the possibility to modularize your business flows by outsourcing and reusing certain parts of the main flow as subprocesses. When a running process enters a call activity, a new process instance of the specified process is started, while the main execution is put on hold. As soon as the called process has finished, the control goes back to the parent.

In Flows for APEX 22.2 call activities can be used to include the call of another diagram within a workflow and by that, execute this as a subprocess within the main process.

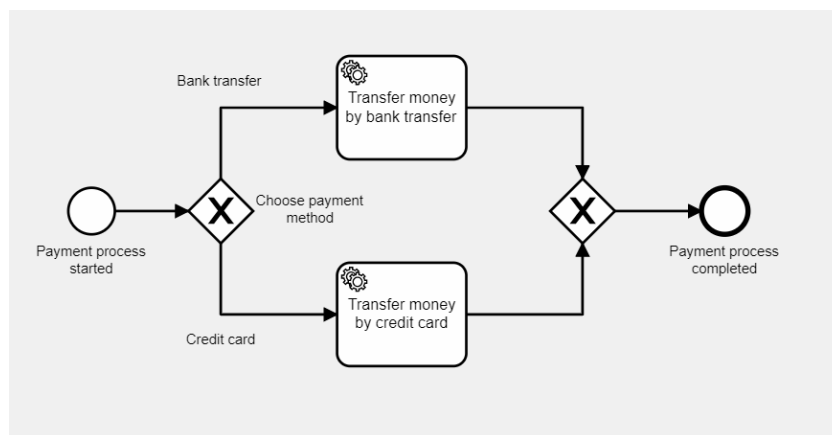
Open the **Flows for APEX App** and create a new Process by clicking **Create Model** on the *Flow Management* page. Enter *Payment* as the **name** and click the **Create** button. Open the new flow in the Flow Modeler afterwards by clicking **Modify Diagram**.

This flow is designed to be a substitution for the prior *Make payment* script task to bring additional logic to the payment process while keeping modularity.

Model the new flow as depicted by creating the needed objects and keep the naming structure from the previous examples. For the sake of convenience, we will use the same function call we used before on both *Service Tasks*, so enter

```
tuto_expense_pkg.make_payment;
```

in the associated fields for the **PL/SQL Code**.



If you include flows in another (parent) flow, you often may want to transfer data from the calling process to the called process. In Flows for APEX, this is referred to be *In and Out Variables*. Select the process object of the model by clicking somewhere on the empty canvas around your placed objects. In the **General** tab of the properties panel, you can see the attribute **Is Callable**, which defaults to *No*. Set this to *Yes* to show the **In/Out Variables** tab. In this tab you can specify, which variables you'd expect to be passed from a calling process as well as which variables this process should return after completion.

Add a new *In Variable* by clicking the '+' button to open the details below. Enter *PAYMENT_METHOD* as the **Name** and keep the **Data Type** setting unchanged. Under **Description** you can provide a help text of this variable to indicate users calling this diagram, how this variable is used inside the process. In this example, just enter *Method to proceed the payment*.

For the exclusive gateway we want to use *Routing expressions* again and therefore use this *In Variable*. Select the **Bank transfer** path, select *Expression* as **Condition Type** and enter the following **Condition**:

```
:F4A$PAYMENT_METHOD = 'BANK_TRANSFER'
```

In the same way, specify the **Credit Card** path by using the **condition**:

```
:F4A$PAYMENT_METHOD = 'CREDIT_CARD'
```

The screenshot shows the configuration interface for a process named 'Process_v5bas4kx'. The 'In/Out Variables' tab is active. Under 'In Variables', there is a list containing one entry: 'PAYMENT_METHOD : Method to proceed the payment'. Below this, the 'Out Variables' section is empty. At the bottom, the 'Variable Details' section shows the variable name 'PAYMENT_METHOD', the data type 'Varchar2', and the description 'Method to proceed the payment'.

Save your changes to the flow and return to the **Flow Management** page. Open the first model *ExpenseFlow* now.

Now we need to change the *Make payment* script task to call the new flow instead. Select the task and click on the **wrench** icon next to it to open the task type list. Switch its type to *Call Activity*. In the properties panel you can see the call activity relevant attributes now.

In the *General* tab, you can specify the called diagram in the *Called Subprocess Diagram* section. Leave the **Input** set to *Use Flows meta data* and select the *Payment* diagram below in the **Called Diagram** list. The **Versioning** attribute can be left at *Latest version*, so each time you update the payment diagram, it automatically gets the newest version on starting a new process instance.

Now open the **In/Out Mapping** tab to set the needed variables. You can click on **Load defined variables** to load the variables sets from the Flows for APEX metadata. The variable *PAYMENT_METHOD* gets automatically added to the *In Variables* list, including name, data type and the description text provided. After selecting the variable from the list, you can specify the explicit value for that value using the *Variable Expression* attributes.

In a real business case, you would probably add another item for the payment method somewhere in the user forms, use this to set a variable in the parent diagram and copy that value here to the called diagram's variable. For testing purposes, we will assign a *Static Value* here, so the payment method will be the same on all instances. Enter either *BANK_TRANSFER* or *CREDIT_CARD* as the **Expression** and save your changes to this model.

Make_payment

General In/Out Mapping

Load defined variables

Copy business reference

In Variables

PAYMENT_METHOD : BANK_TR

Out Variables

Variable Details

Name

PAYMENT_METHOD

Data Type

Varchar2

Method to proceed the payment

Variable Expression

Expression Type

Static

Expression

BANK_TRANSFER

Static value

We don't have to make any changes to the application here as the process flow for the user remains untouched. When clicking through the process, you can look at called diagrams in the **Flow Monitor** by opening the instance and clicking on the '+' icon on the call activity in the viewer afterwards. For navigating back to parent diagrams, you can use the breadcrumb bar at the top of the canvas, which will show the current call tree. This comes especially handy when you have multiple levels of nesting in your business flows.