Appian
Academy

Appian Step-by-Step #1

Exercise to Accompany
Welcome to the Appian Developer Learning Path

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Introduction

How to Complete the Exercises in this Guide

This guide offers a series of exercises that will teach you how to build a functional Appian application, step-by-step. Please use this guide to support your learning throughout the Appian Developer online learning path.

To understand specific Appian concepts and features, we recommend that you first view our online courses (access our curriculum at academy.appian.com) and then complete the associated exercises before moving to the next course in the path.

What Will I Build?

You will build the Acme Exercise Application (AX). This application automates the process of registering a new vehicle in the fleet for the fictional company Acme Automobile, Inc. The app is designed for three personas: Registrars, Supervisors, and the Maintenance Crew (Mechanics). Registrars use the AX app to add new vehicles to the fleet, Supervisors review and approve new vehicles, and the Maintenance Crew requests maintenance and keeps records for each maintenance project. After you build all key design objects, you will assemble them into a user-friendly site that Registrars, Supervisors, and the Maintenance Crew will use for adding and approving vehicles and for accessing all fleet-related reports.

You will build most objects in this exercise. However, there will be some selected data and objects that come from the existing Acme Automobile Reference application (AA). You can also use the Acme Solution (AS) application as a reference tool. Review this application to see how specific objects or fields are configured, or test the app to see how the features work from a business user's perspective. You can find this app in your Appian Community Edition practice environment.

How Can I Practice?

Sign up for Appian Community Edition here: https://community.appian.com/p/my-learning-journey/. This free, unlimited-time Appian instance comes with the pre-built AS application that you can use to reference while building your own version.

If you already have access to a practice environment, you will need a user account with either System Administrator access, or an account for a Basic User who is a member of the Designers and Process Model Creator groups. Request access from your site's System Administrator.

Use Initials in a Shared Environment

If you work in a shared environment, prefix all application objects with your initials to avoid naming conflicts. For example, if your initials are JJ, name your group JJ AXA All Users.

Save Often

Appian does not automatically save updates, so save your objects frequently.

Additional Resources

Appian provides a plethora of training resources for Appian developers. The following resources are particularly popular with our learners:

Academy Online - Appian's online courses provide useful survey courses, step-by-step tutorials, and some additional practice exercises. Explore these resources at your own pace. Survey courses will help you with developing a better grasp of the range of topics you need to learn about. Video and print tutorials will help you get hands-on experience with Appian.

Appian Documentation - Appian's product documentation will provide you with the overview of key Appian features, newest release information, additional tutorials, and helpful patterns and recipes to implement in your app.

Community Discussions for New Users - Join our community of experts to ask questions and find answers from past discussions.

Exercise 1: Welcome to the Appian Developer Learning Path

Access ACE Environment

To complete the exercises in the Appian Developer learning path, sign up for access to the Appian Community Edition (ACE) environment. Once you sign up for the environment, you will have access to a free, unlimited-time instance. The environment comes with pre-loaded objects that you will need as you go about building your ACME Automobile application. Follow the steps below to access the ACE environment.

1. Navigate to the Appian Community website, and log into your Appian Community Edition account.
2. Click the **Learn** tab.
3. Click the **My Learning Journey** icon.
4. Click **Get my site** in the **My Community Edition Site** card.
5. Choose your region, and generate your site.

When an instance is assigned to you, your instance URL is added to the My Community Edition site card on the My Learning Journey page. Your username, which is the email address associated with your Appian Community account, and password is sent via email. If you do not receive your site's credentials, please email CommunityEdition@appian.com for help.

Download and save these exercises as a reference.

Appian
Academy

Appian Step-by-Step #2

Exercise to Accompany
Create an Application: First Steps

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 2: Create an Application

Create an Application

You will start building your app by first creating the application itself. This object is simply a container for design objects specific to this application. You will use it for keeping your objects together, and for packaging them into a zip file before you deploy the app to a different environment. Follow the steps below to create an application.

1. Log into your **Appian Designer** environment.
2. Navigate to the **Appian Designer** by clicking the

   Navigation button  next to your avatar and selecting **Appian Designer**. You can also simply change the URL from **/suite/site** to **/suite/design** in the address bar of your browser.
3. In the **Appian Designer**, click **New Application**.
4. In the **Create New Application** dialog box, retain the **Create from Scratch** option. Name your app **Acme Exercise**, add a simple description: "An application for managing vehicles and maintenance in a fleet," and leave the prefix as **AX**. Click **Create**.
5. After you click **Create**, you will be prompted to configure security for the app. Because you don't have application groups yet, simply click **Save**. You will come back to app security later.

After you create the new application, it will open automatically.

Create Parent Folders

In this exercise, you will create folders that will help you organize and secure application objects. In Appian, there are four types of folders: Rule, Process Model, Knowledge Center, and Document. Follow the steps below to create parent folders.

1. In your application, click **New**, and select **Folder** from the dropdown list.
2. Create the **Rule** folder first. This will be the parent folder for other rule folders in the application. Choose the **Rule** option under **Type**, name your folder **AX Rules**, and give your folder a description: "Contains all constants, expression rules, and interfaces used in the AX app." Leave the parent folder field blank. Click **Create**.
3. Ignore the security warnings for now. You will revisit security settings later after application groups are created. Click **Save**.
4. In the same manner, create the following parent folders: **AX Process Models** and **AX Knowledge Center** (folder for application documents). Remember to choose the correct type for each folder.

Create Sub-Folders

Sub-folders are folders that reside in parent folders and inherit the security of the parent folder. Follow the steps below to create sub-folders.

1. Create three sub-folders in the **AX Rules** parent folder: **AX Constants**, **AX Expressions**, and **AX Interfaces**.

    - For each one, click **New**, and select **Folder** from the dropdown list.
    - Name your folder, add a simple description, and select the **AX Rules** folder in the parent folder field.

2. Alternatively, you can create all folders from within the AX Rules folder. Simply access AX Rules, and create all sub-folders from within this parent folder. With this method, the parent folder field will be pre-populated with AX Rules.

3. From within the **AX Knowledge Center** parent folder, create the sub-folder **AX Documents**. Make sure that the parent folder field contains the **AX Knowledge Center.**

Appian
Academy

Appian Step-by-Step #3

Exercise to Accompany
Manage Users and Groups

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 3: Manage Users and Groups

Manage Users

When you signed up for the Appian Community Edition (ACE) environment and launched your site, a user with your credentials was created for you. For this exercise, you will not need to create additional users. As you build your application, add and remove your user to the different groups to test the security of your application features.

Learn more about creating and managing users by visiting Appian Documentation.

Create Groups

In this exercise, you will create the groups for your application: AX All Users, AX Administrators, AX Supervisors, AX Registrars, and AX Maintenance. Groups are important because they are used to set security to the application and its objects. The presence of All Users and Administrators groups is standard and considered a best practice in most apps. The All Users group includes all users with access to the app, and it is always the parent group for all other groups in an application. The AX Administrators group includes all users administering the app and its objects. The AX Supervisors and AX Registrars groups include business users who will be accessing the app to complete their work. These groups will help you define the access to specific app features for your business users.

As you create groups and other objects, remember to use the app prefix in all names. This will become helpful in scenarios where you have to discern between similar objects from different apps. Follow the steps below to create the groups.

1. In your application, click **New** and select **Group** from the dropdown menu.
2. In the dialog box, name the group as **AX All Users**, and add a description: "Parent group for all roles in the AX application."
3. Leave the Parent Group and Group Members fields empty since this group does not have a parent or group members yet. Change the Visibility to **Public**, and click **Create**.
4. You will be prompted to review group security. Wait on the security review and setup until you create the AX Administrator group. Click **Save**.
5. Once the AX All Users group is created, click the group name to access it.
6. From within **AX All Users**, follow the steps above to create the remaining groups: **AX Administrators**, **AX Supervisors**, **AX Maintenance**, and **AX Registrars**. Add the **AX All Users** group as their parent group. Do not set security for these groups. You will secure them using their parent folder, AX All Users.
7. Add yourself to **AX Administrators, AX Supervisors**, **AX Maintenance**, and **AX Registrars**. You can add a user by drilling into each target group and clicking **Add Members**. This way, you will be able to see the features designed for specific users.

You can also remove yourself from a group to test the security of these features.

Configure Group Security

After you have the two main app groups AX All Users and AX Administrators, you can start securing all application objects. This is what we call object-based security in Appian. You will start by securing groups first. To do so, you will secure the AX All Users parent group. All other groups will inherit security settings from its parent. Follow the steps below to set group security.

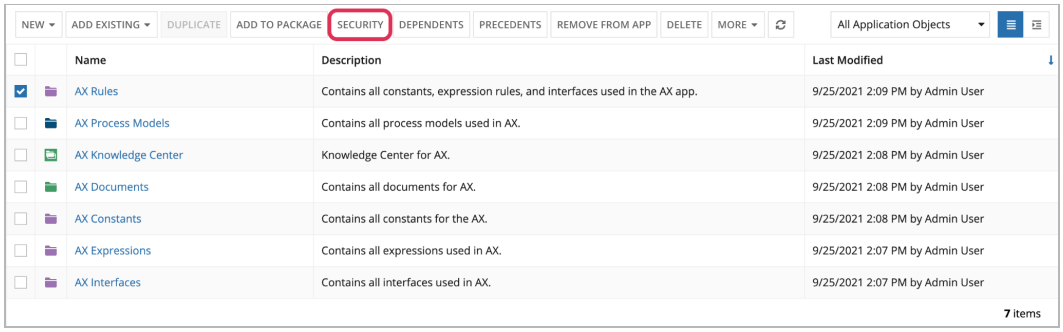
1. From within the **AX All Users** group, click the **Settings** button  for this group, and then select **Security** from the dropdown menu.
2. In the dialog box, click **Add Users or Groups**. Start typing **AX Administrators** in the User or Group field, and then select this group from the list of suggestions. In the **Permission Level** field, retain **Administrator**. Click **Save Changes**.

Now that we set security to the AX All Users group, all other child groups inherit this security setting. This is what we call security inheritance in Appian.

Configure Folder Security

In Exercise 2, you created parent folders for your application objects. Now that you have your application groups, you are able to configure folder security. Documents or rules contained in a parent folder can automatically inherit user rights from the parent folder. Process Model folders do not provide security inheritance. Follow the steps below to add security settings to your parent folders.

1. Select **Folder** from the **Object Type** filter on the left side of Appian Designer.
2. Check the box next to **AX Rules**, and click **Security**:



3. Click **Add Users or Groups**.
4. In the **User or Group** column, type and select **AX Administrators**. Set the permission level to **Administrator**.
5. Click **Add Users or Groups**, and add **AX All Users**. Set the permission level to **Viewer**.

6. If your individual user is selected as an Administrator, remove your user from the list. Appian recommends only using groups to configure security. Click **Save Changes**.

7. Repeat this process for **AX Knowledge Center** and **AX Process Models**.

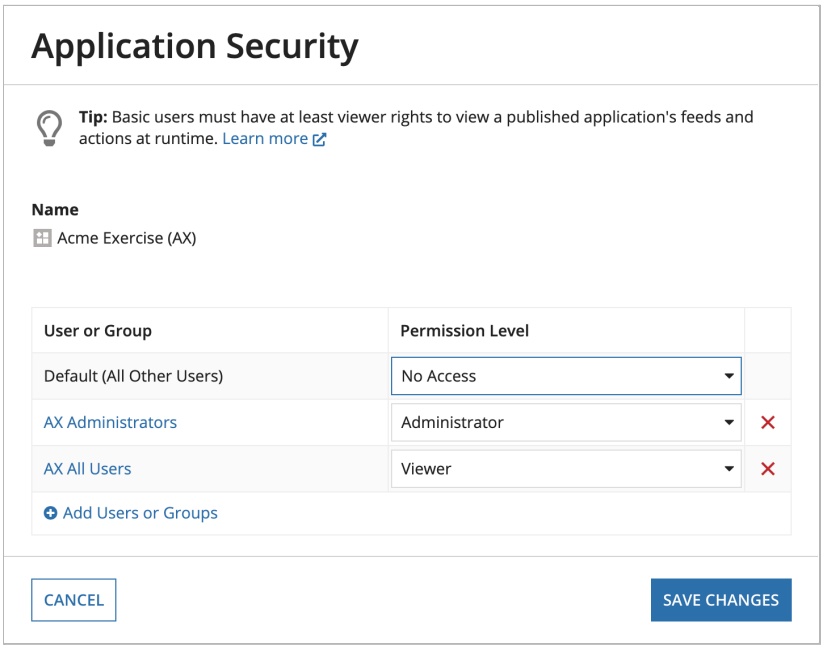
Configure Application Security

Now that you have the groups, you will set security to the whole AX app. From now on, you should also set security on all objects that you create. Follow the steps below to set security to your application.

1. In **Appian Designer**, select the checkbox next to **Acme Exercise (AX)**, and click **Security**.

2. In the **Application Security** dialog box, click **Add Users and Groups**, and type **AX Administrators** in the **User or Group** field. Select the **Administrator** permission level.

3. Click **Add Users or Groups** again. In the **User or Group** field, type **AX All Users**, and keep the default **Viewer** permission level:



4. If your individual user is selected as an Administrator, remove your user from the list.

5. Click **Save Changes**.

Appian
Academy

Appian Step-by-Step #4

Exercise to Accompany
Expressions: Transform Your Data

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 4: Expressions

In this exercise, you will create two types of Rule objects that you need for your application. Specifically, you will create six constants and one expression rule.

Create Constants

A constant is a literal value that can be called from any expression, and they can be reused across multiple objects throughout your application. In this section, you will create constants that point to the groups and folders you just created. Once you create a constant that points to a group or folder, you will be able to call it from any expression. Follow the steps below to create a constant.

1. From within the **AX Constants** folder, click **New**, and select **Constant** from the dropdown menu.
2. Type **AX_ADMINISTRATORS_POINTER** for the **Name**, and add a description: "A pointer to the AX application administrators group."
3. In the **Type** field, select **Group**.
4. In the **Value** field, type **AX Administrators**, and then click **Create**.
5. In the same manner, create constants for the remaining four groups: **AX All Users, AX Supervisors**, **AX Registrars**, **and AX Maintenance.**
6. Now, you will create a constant that points to the AX Documents folder. Click **New**, and select **Constant** from the dropdown.
7. Name the constant **AX_DOCUMENTS_FOLDER_POINTER**, and provide a description: **"**Points to the AX Documents folder**."**
8. Select **Folder** for the **Type** field, **AX Documents** for the **Value** field, and **AX Constants** for the **Save In** field.
9. Click **Save**.

Create Expression Rule

An expression rule is a statement that evaluates to return a v alue. In this section, you will create an expression that will help format a user's name. This expression will be used in a later exercise to help you format the name field correctly. Follow the steps below to create this expression rule.

1. From within the **AX Expressions** folder, click **New**, and select **Expression Rule** from the dropdown menu.
2. Type **AX_DisplayUser** for the **Name**, and add a description: "Formats a user's name for the AX app." Click **Create**.

3. In the **Rule Input**s pane, click the **plus sign** to add a rule input:

    - **user** (Text)

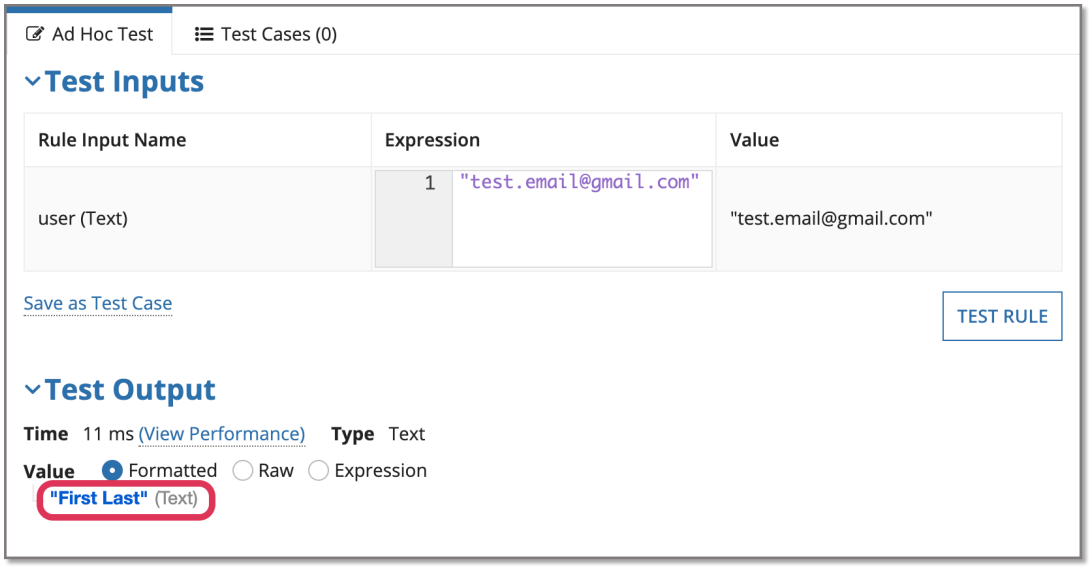
4. Enter the following expression into the **Expression Editor** on the left:

```
proper(
  if(
    isnull(ri!user),
    "",

    isusernametaken(ri!user),
    user(ri!user, "firstName") & " " & user(ri!user,
"lastName"),

    joinarray(split(ri!user, "."), " ")
  )
)
```

    This expression first checks whether the user's username is null or not. If the user's username is not null, the expression then checks whether the input is an active user or not. Since the process for formatting users and non-users is different, the last two lines in the expression are both ways to reformat the input as First and Last Name.

5. Click the **Format** icon in the toolbar to automatically format your expression.

6. Click **Save Changes** (or press ctrl/cmd + s). Test this expression by typing your ACE email (wrapped in quotes) in the **Test Inputs Expression** field. Click **Test Rule** to see the formatted name:



Appian
Academy

Appian Step-by-Step #5

Exercise to Accompany
Design Appian Records Part 1: Accessing Your Data

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 5: Records Part 1, Accessing Your Data

In this exercise, you will create a record for displaying all relevant information about the vehicle fleet to business users. Creating a record will involve several incremental steps:

1. Create the **AX Vehicle Record**.
2. Create a **Custom Record Field.**
3. Configure the **AX Vehicle Record List**.
4. Add a **User Filter** to allow users to filter the fleet by the mileage category.
5. Create a **Summary View Interface** to display the info about each vehicle.
6. Connect this interface to the **Summary View** in the AX Vehicle Record.
7. Create an **Action** for adding vehicles to the fleet.

Follow the steps below to create the vehicle record, and to set up all of the record features.

Create the Vehicle Record

1. In your application, click **New**, and select **Record Type** from the dropdown menu.

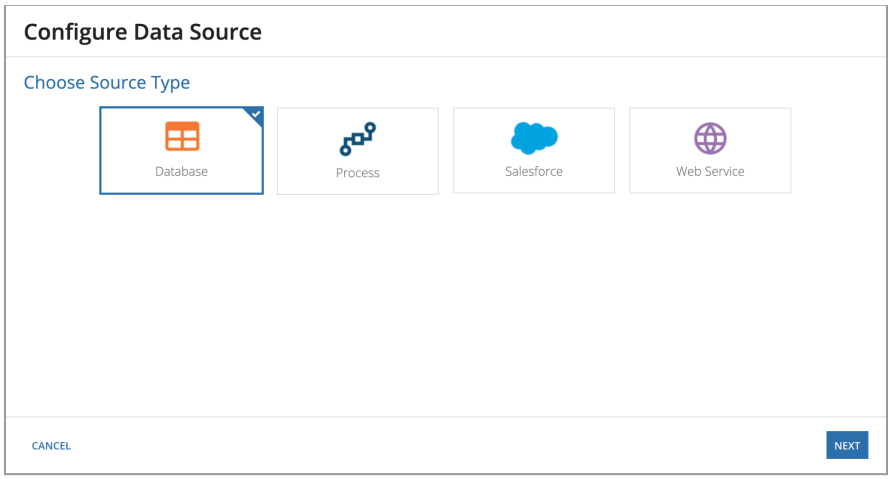
2. Name it **AX Vehicle**, and type **AX Vehicles** in the **Plural Name** field. The plural name is what business users will see in their Site, so pick the name that will make sense to your business users. Note that application prefixes are not typically standard for record names. You are using a prefix for this exercise because a Vehicle record already exists in the AA Reference Application.

3. Add the description: "The list of vehicles managed by the AX application." Click **Create**.

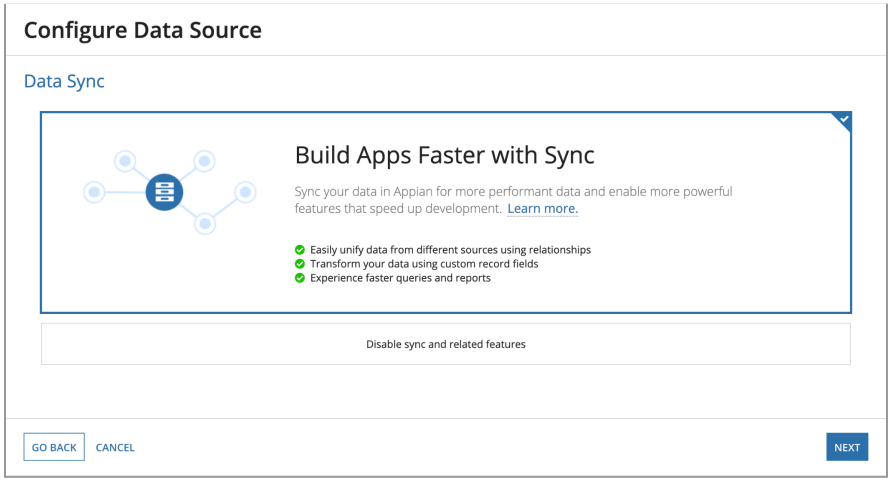
4. In the next dialog, configure the security of the new record. Add **AX Registrars** and **AX Supervisors** as **Viewers**. Add **AX Administrators** as **Administrators**.

5. If your individual user is selected as an Administrator, remove your user from the list. Click **Save**.

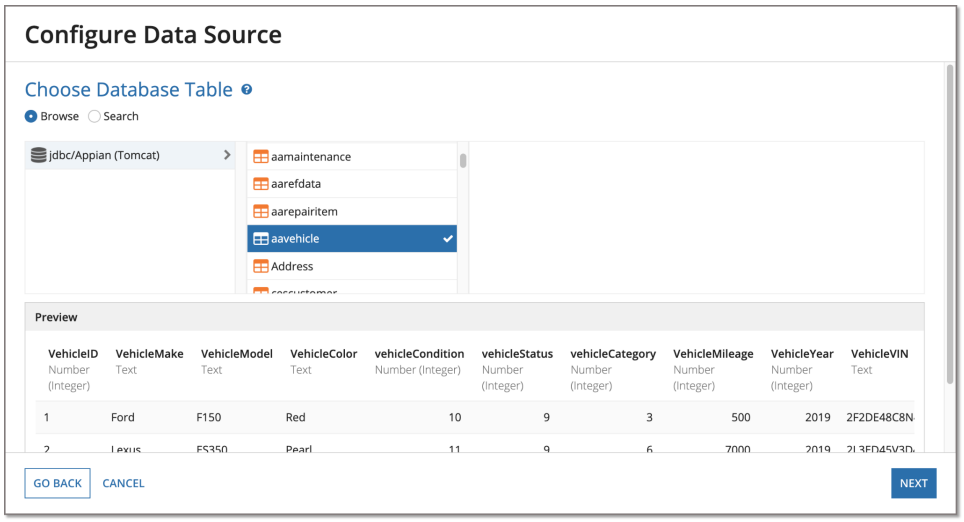
6. Click **Tell Us About Your Data**. In the **Configure Data Source** dialog, ensure that **Database** is selected, and click **Next**.



7. Select **Build Apps Faster with Sync**. Click **Next.**



8. Under **Choose Database Table**, select **aavehicle**. Click **Next**.

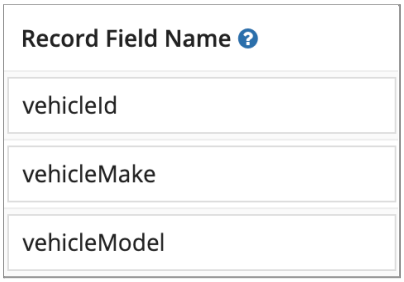


9. Let's add a source filter to exclude vehicles that are more than ten years old. In the next dialog, click **Add Filter**, and set the following configurations:

    - Select **VehicleYear** as the **Field**.
    - Select **>** as the **Condition**.
    - Enter **2011** as the **Value**.

    Click **Next**.

10. Preview the fields, and edit the Record Field Names to use camel casing. Camel casing refers to the practice of writing phrases without spaces and punctuation, where the separation of words is indicated with a capitalized letter (for example, vehicleStatus or maintenanceStartDate):



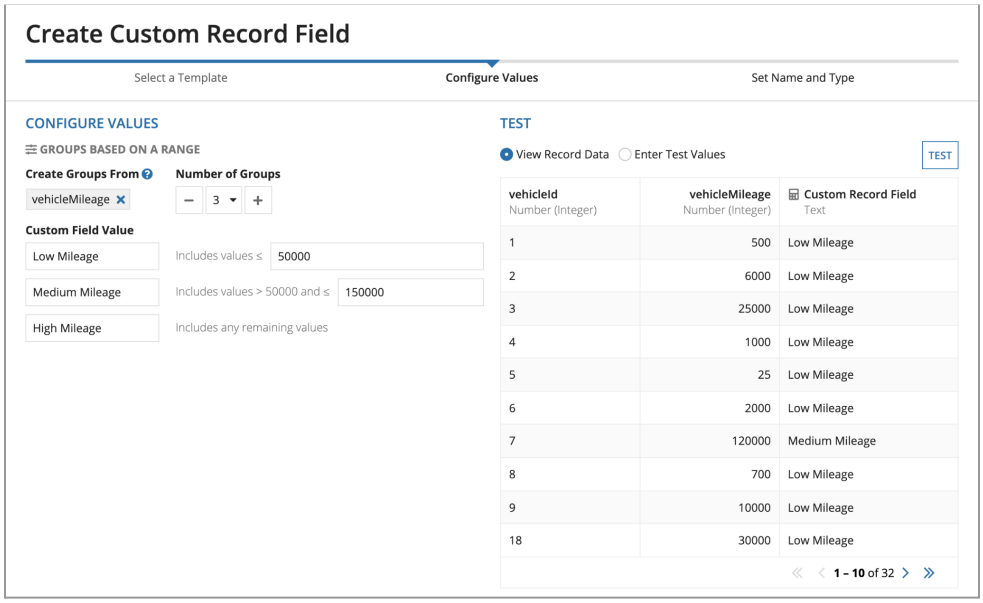
    Click **Finish**.

11. Before you build additional features, save the record by clicking **Save Changes**, and check how the record looks from the business user perspective. Click the link located next to the record name:

Create a Custom Record Field

Before you configure the record list, you will first create a custom record field. For your app, you will create a new custom record field that displays a vehicle's mileage category: Low Mileage, Medium Mileage, or High Mileage. You will add this new custom record field to your record list in the next section. Follow the steps below to create a custom record field.

1. Click **Data Model** in the left menu, and then click **New Custom Record Field**.
2. In the **Select a Template** section, click **Groups Based on a Range**. Click **Next**.
3. In the **Configure Values** section, select **vehicleMileage** in the **Create Groups From** field.
4. Set the **Number of Groups** field to **3**.
5. Name Group 1 **Low Mileage**, and set the Upper Limit field to **50,000**. Name Group 2 **Medium Mileage**, and set the Upper Limit field to **150,000**. Name Group 3 **High Mileage**.
6. Click **TEST** to preview how the custom record field will display:



Click **Next**.

7. In the **Set Name and Type** section, update the name of the custom record field to **mileageCategory**.

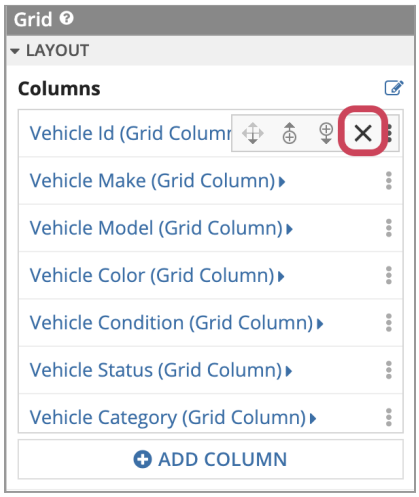
8. Click **Create**, then **Save Changes** to save the record.

Configure the Record List

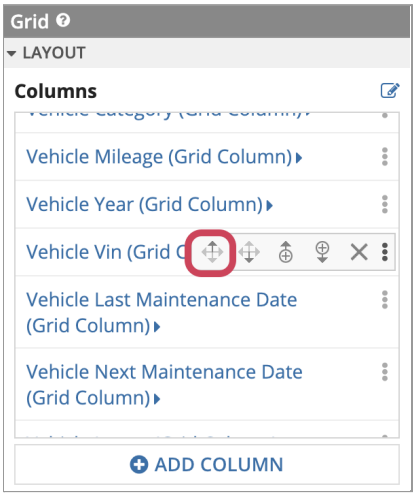
Next, you will edit the columns of the vehicles grid to display the following columns: Vehicle VIN, Make, Model, Year, Next Maintenance Date, Status, Added By, and Image. You will sort the grid by the vehicle make, condition, category, and status, and you will add a clickable link to the vehicle VIN column. Later, you will connect this clickable link to an interface with the relevant information for each vehicle. Follow the steps below to configure the record grid.

1. Click **List** in the left-hand menu, then click **Edit List**.

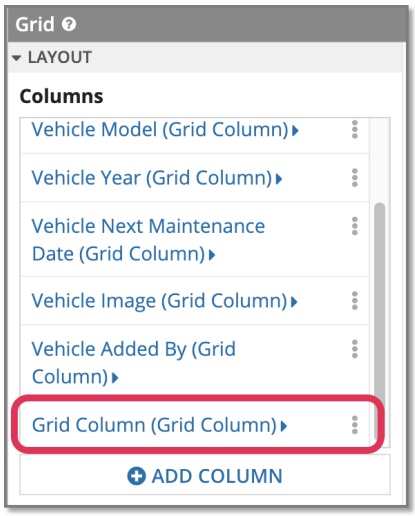
2. In the **Edit Record List** dialog, review the list of fields. You will keep only these fields: VIN, Make, Model, Year, Next Maintenance Date, Added By, and Image. To delete all other fields, click the in-line **X** buttons next to those fields:



3. Move the **VIN column** up so that it is the first column in the grid. Use the in-line **arrow** to move this column.



4. Click **Add Column**, and select the newly created **Grid Column** link to drill into the column.



5. Update the label name to **Mileage Category**.
6. Select **mileageCategory** in the **Sort Field** field**.**
7. Select **mileageCategory** in the **Display Value** field.
8. Move the **Image** column so that it is the last column in the grid. Use the in-line **arrow** to move this column.
9. To simplify the column names, drill into each column, and remove "**Vehicle"** from all of the column labels.

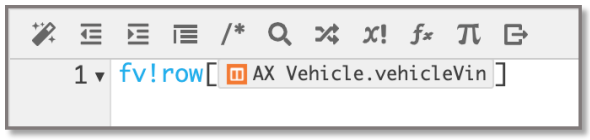
Edit Record List Columns

Now that you have the columns you need for your record list, you can start editing individual record list columns.

1. Transform the **VIN field** into a clickable link:

    - Select **VIN**, then click the **Display Options** button.
    - In the **Display Options** dialog, select **Record Link**.
    - Under **Display Value**, click **Link**.
    - Under **Links**, click **List of Links**.
    - Click **Record Link (Record Link)**.
    - Click the **Expression Editor** icon next to **Label**, and delete all text in the **Expression Editor**.
    - Type **fv!** and then select **row**.
    - Type square brackets **[ ]**, enter **recordType!** and then select **AX Vehicle**.
    - Type a period, and then select **Fields** and **vehicleVIN**.
    - Click **OK**. Your expression will look like this:

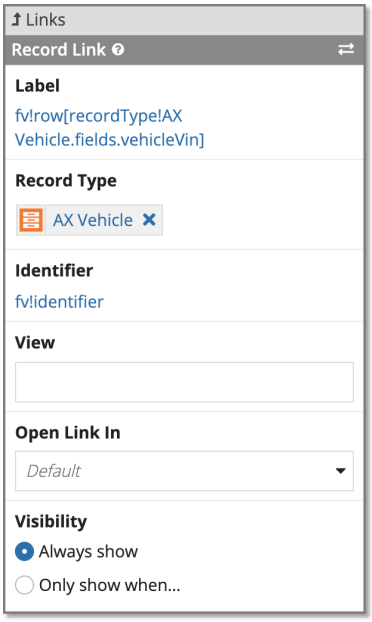
        ```
        fv!row[AX Vehicle.vehicleVin]
        ```

        Click **OK**.

    - Back in the **Record Link** section, click the **Expression Editor** icon next to **Record Type**.
    - In the **Expression Editor**, delete all text.
    - Type **recordType!**, and then type and select **AX Vehicle**. Click **OK**.
    - Click the **Expression Editor** icon next to the **Identifier** field.
    - Delete the expression.

- Type **fv!** and select **identifier**. Click **OK**:



2. Remove the link from the **Make** column by following the steps below:

   - Under **Columns** in the left pane, click **Make**.
   - Under **Display Value**, click **Clear** to remove the link.
   - Select **vehicleMake** from the dropdown.

3. Format the **Added By** column by following the steps below:

   - Under **Columns** in the left pane, click **Added By**.
   - Click the **Expression Editor** next to **Display Value**. You will call in the expression rule you created in Exercise 4. To use the expression rule, clear the existing text, and enter the following:

     ```
     rule!AX_DisplayUser(fv!row[recordType!AX
     Vehicle.fields.vehicleAddedBy])
     ```

     rule! is a type of object prefix. Use **rule!** to indirectly reference an expression rule or interface. **fv!row** is a type of function variable. This function value sets the display value of every column and contains all the data for the entire row. Notice how the names in the Added By column are now formatted correctly.

4. Format the **Image** column by following the steps below:

   - Under **Columns**, click **Image**.
   - Under **Display Value**, click **Display Options**.
   - Select **Document Image**.
   - Under **Display Value**, click **Image**.

- Under **Images**, click **Document Image**.
- Under **Document**, click **a!EXAMPLE_DOCUMENT_IMAGE()**, and clear everything in the expression editor.
- Enter the expression:

```
fv!row[recordType!AX Vehicle.fields.vehicleImage
```

- Click **OK**. You will see the vehicle images appear.

5. Adjust the alignment and width for the columns.

    - Click on each column and scroll to the **Alignment** and **Width** fields.
    - Adjust the columns as needed. We suggest following our best UX practices by keeping the text left-aligned and numbers right-aligned. Leave the default setting for width.

6. Click **OK**, and then **Save Changes**. Remember that if you want to review the updated grid as it appears to the business user, you can do so by clicking the link located next to the record name.

Add a User Filter

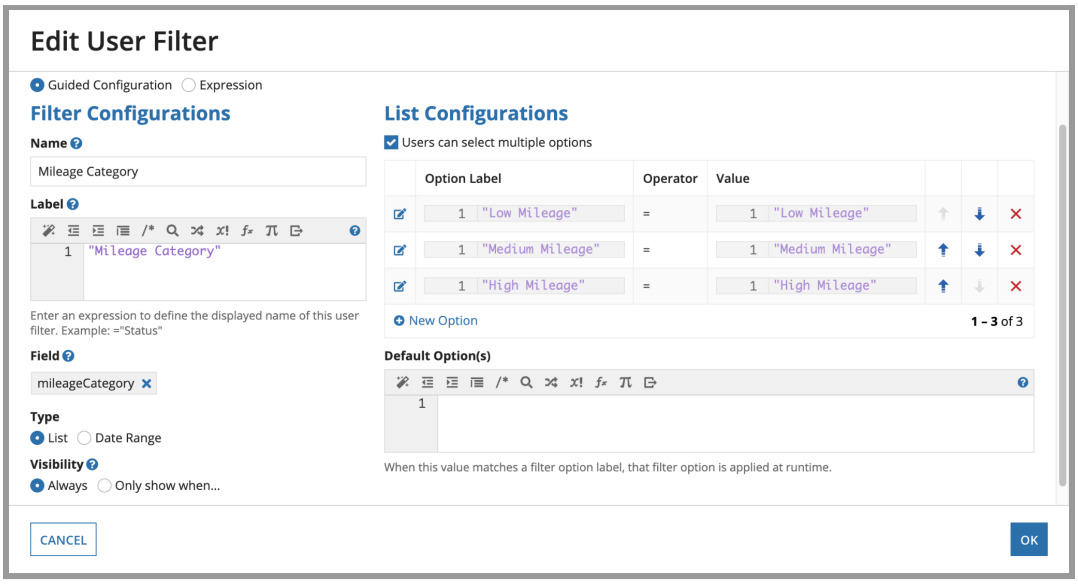
You will now add a user filter to the record list to allow business users to filter for vehicles by mileage category. Follow the steps below to build this user filter.

1. In the **AX Vehicle** record, select **User Filters** in the left navigation. In the **User Filters** section, click **New User Filter**.

2. Select **Create New Filter**, and configure the following items:

    - Type **Mileage Category** in the **Name** field.
    - Type **"Mileage Category"** in the **Label** field. Note that you are using quotation marks because it is an expression.
    - Select **mileageCategory** from the **Field** dropdown.
    - Click **New Option**, and type **"Low Mileage"** in both the **Option Label** and **Value** expression editors. Leave **=** in the **Operator** field, and click **Save Filter Option**.

- In the same manner, create the **Medium Mileage** and **High Mileage** options:

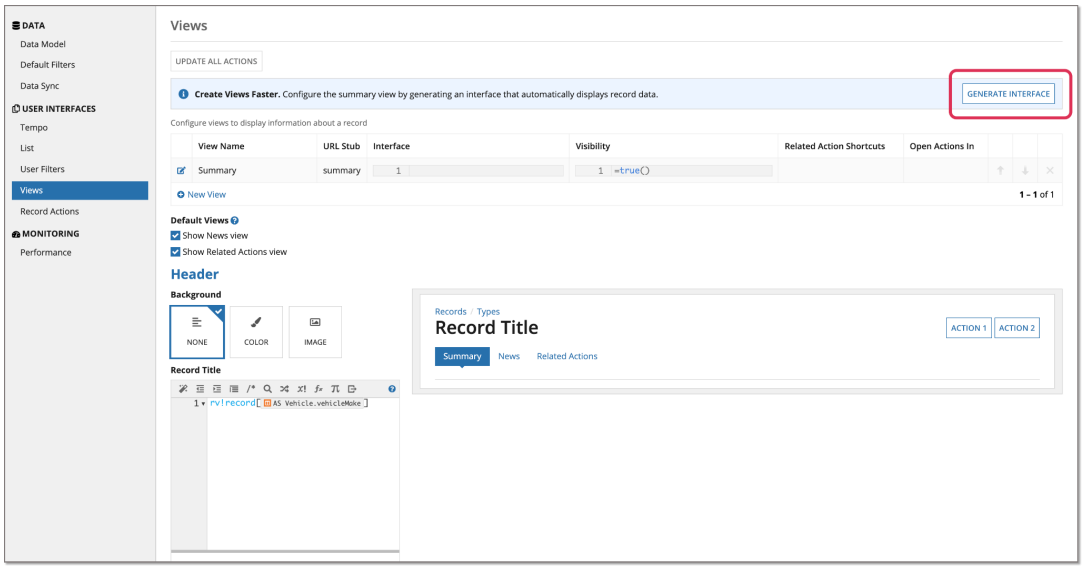


- Click **OK**, then **Save Changes**. Click the link located next to the record name to test the user filter.

Generate the Summary View Interface

In this section, you will create an interface to display the summary vehicle information in the AX Vehicle record. Later, you will link this interface to the vehicle grid. Once business users review the vehicle grid, they will be able to click the VIN link to access additional vehicle information. Follow the steps below to build the summary interface.

1. From the **Views** page of the **AX Vehicle** record, click **GENERATE INTERFACE**.

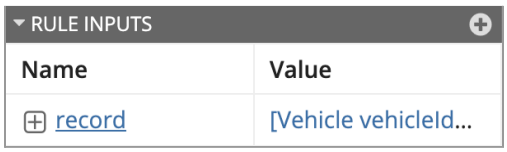


2. Name the interface **AX_VehicleSummary** and add a description: "A reusable section to display the read-only vehicle details."

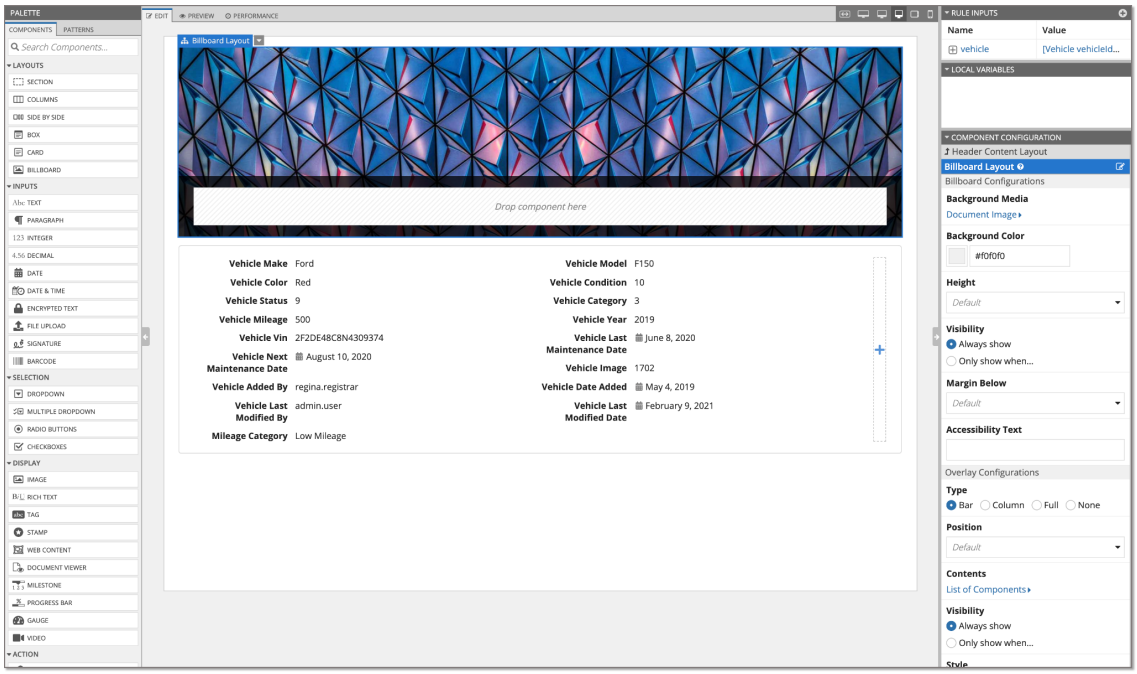
3. Select the **AX Interfaces** folder in the **Save In** field. Click **Create**.

4. Once the interface is generated, click the **record** link in the **Rule Inputs** pane.

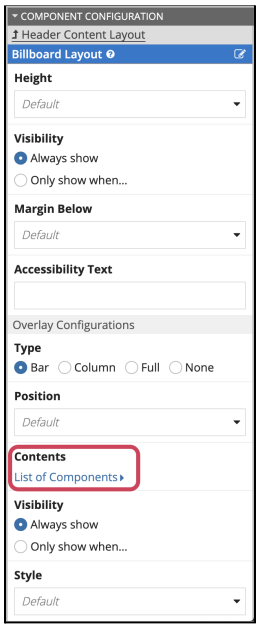
5. Replace the rule input name with **vehicle**. Click **OK**.



6. Use the **Components Palette** to find components that you can use in your interface. Drag and drop the **Billboard** component above the vehicle details:



7. Click the **Billboard Layout** in the canvas to make sure it is selected. In the **Component Configuration** pane for the Billboard Layout, scroll down to **Contents**, and click **List of Components**:



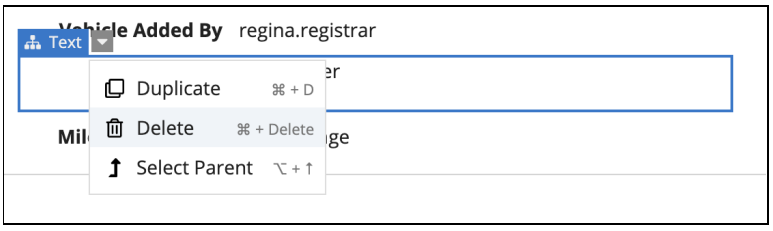
8. Click **Add Component**, and select **Rich Text**. Click the **Rich Text** link.

9. In the **Display Value** editor, type **Vehicle Information**. Using the editor toolbar, change the size to **Medium Header**. We suggest following our best UX practices by using varied font features to highlight the page title.

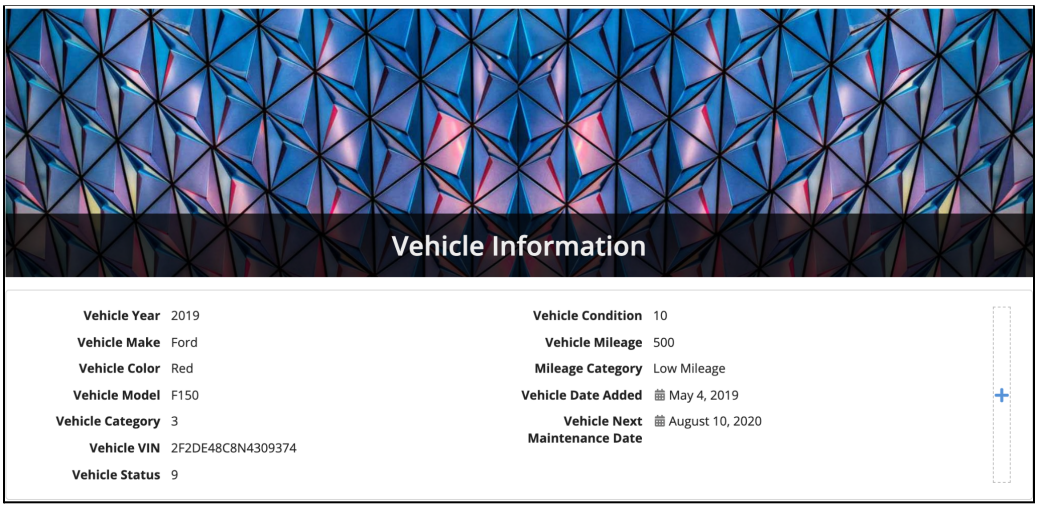
10. Under **Alignment**, select **Center** from the dropdown menu.

11. Select the **Card** layout with the vehicle details, and configure the following items:

    - Remove the following fields: **Vehicle Last Maintenance Date**, **Vehicle Last Modified Date**, **Vehicle Added By**, **Vehicle Image**, and **Vehicle Last Modified By**. To remove fields, click an **individual text box**, and select **Delete** from the dropdown menu.



- Rearrange the fields by dragging and dropping the text boxes. Rearrange the fields to match the following configuration:



- Remove **"Vehicle"** from the labels. To rename a label, click an **individual text box**, and remove **Vehicle** from the **Label** field.
- Click **Save Changes**.

Add a Vehicle Image

1. Click the **+** next to the second column to create a third column:

2. Drag and drop an **Image** component into the new column.

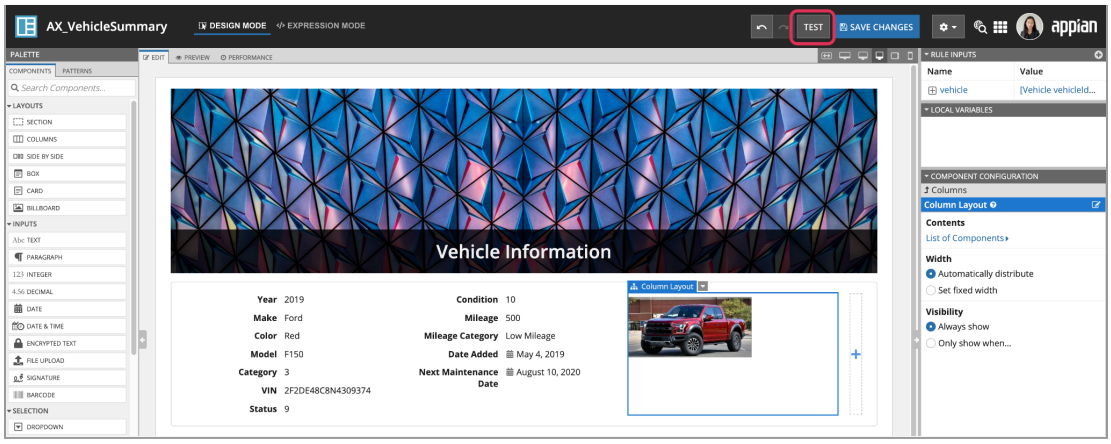
3. Click the **+** in the **Image** field, and select **Document Image**. In the **Component Configuration** pane, click **a!EXAMPLE_DOCUMENT_IMAGE()**. Delete all text in the **Expression Editor**. Type the following expression:

```
ri!vehicle[recordType!AX Vehicle.fields.vehicleImage]
```

   Use the auto-suggestions to populate your expression with the correct data. Click **OK**.

4. You won't need a label for the **Image** field. Delete the **Image** label directly on the canvas. Alternatively, use the **Component Configuration** pane for the **Image** component to delete all text in the **Label** field.

5. Click **Test** to review the test data used to populate the interface.



6. The expression in the Expression Editor uses **a!queryRecordType** to retrieve record data. Click **Set as default test values**, then **Test Interface**.

7. Click **Save Changes**.

Add Interface to Summary View

1. Now that your interface is updated, return to the **AX Vehicle** record, and click **Views** in the left navigation.

2. Click the **Expression Editor** icon next to **Summary**:

3. In the **Expression Editor**, reference the interface you just made by clearing the existing text and entering the following expression:

```
rule!AX_VehicleSummary(vehicle: rv!record)
```

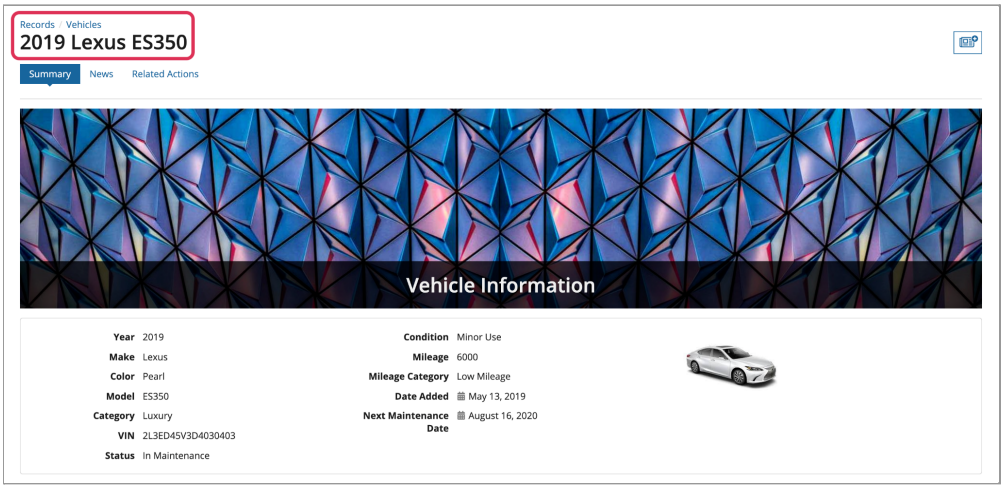
Click **OK**.

4. In the **Record Title Expression Editor**, clear all text, and enter the following expression:

```
rv!record[recordType!AX Vehicle.fields.vehicleYear] & " " &
rv!record[recordType!AX Vehicle.fields.vehicleMake] & " " &
rv!record[recordType!AX Vehicle.fields.vehicleModel]
```

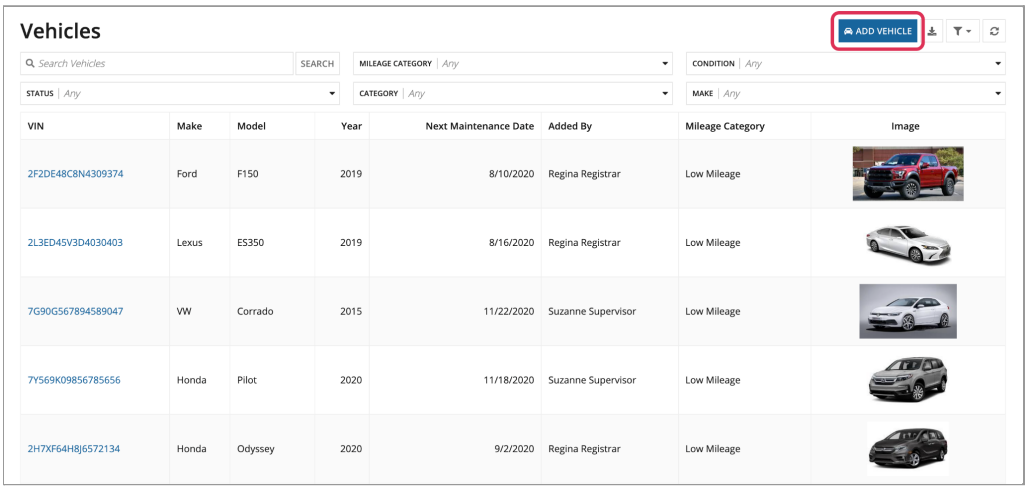
This expression will create a dynamic title for each vehicle summary view. Instead of a generic title, the business user will see the year, make, and model of the vehicle they are viewing. Click **Save Changes**.

5. Preview the vehicle summary view with the new dynamic title. To preview, click the link next to the AX Vehicle record name. Select any vehicle to access its summary view page. Your summary view page should now display a dynamically generated title.



Create a Record Action

In this exercise, you will create a Record Action that will allow Registrars to add a new vehicle to the fleet. A record action will allow Registrars to add a new vehicle by clicking a button from the record list and filling out a form designed for this purpose:



Follow the steps below to set up a record action.

1. In the **AX Vehicle** record, select **Record Actions** in the left navigation.
2. Click **GENERATE A RECORD ACTION**.
3. In the **Choose Your Action** dialog, select **Create**. Click **Next**.
4. Name the action **Add Vehicle**, and add a description, "Action for adding a new vehicle to the fleet." Click **Next**.

5. Review the objects that Appian generates for this action. For this action, Appian generates the following nine objects:

    - Interface for adding a new vehicle
    - Process model for adding a new vehicle
    - Expression rule for creating a vehicle's display name
    - Vehicle Custom Data Type (CDT)
    - Application Data Store
    - Data store entity for Vehicle data references
    - All Users group
    - Rules and Constants folder
    - Process Model folder

6. Make a few changes to the action objects by clicking on the **edit icon** next to each action:

    - Rename AX_createVehicle to **AX_AddVehicle**.
    - Replace AX Users with the existing object **AX All Users**.
    - Replace AX Rules & Constants with the existing object **AX Rules**.
    - Replace AX Models with the existing object **AX Process Models**.

7. Click **Generate Action**. After Appian successfully creates the record action and supporting objects, click **Close**.

8. Click **Save Changes.** Preview the record with the Add Vehicle record action to see how the new feature displays to the business user. To preview, click the link next to the AX Vehicle record name.

Appian
Academy

Appian Step-by-Step #6

Exercise to Accompany
Design Appian Records Part 2: Record Relationships

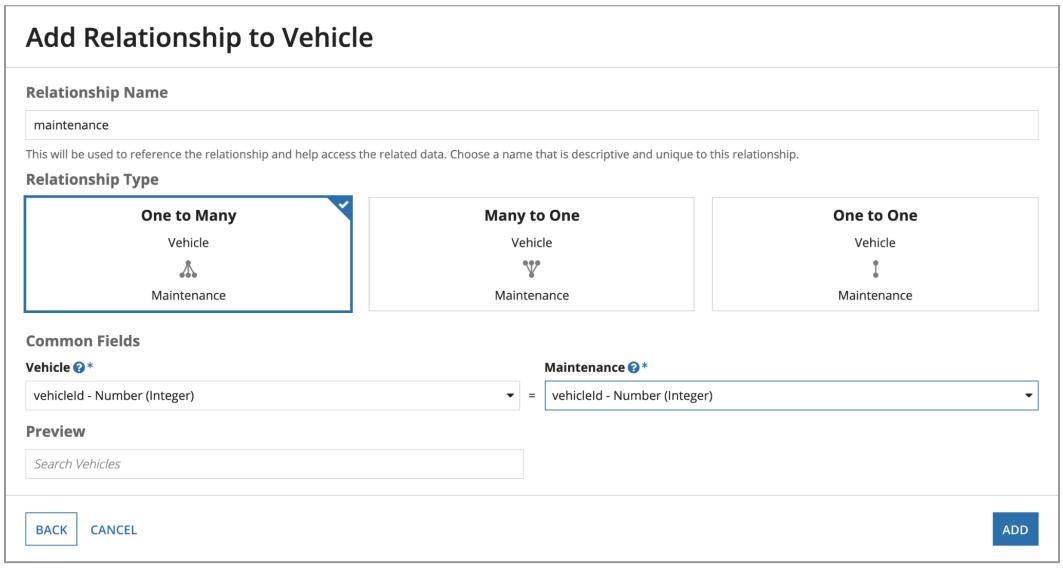
The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 6: Records Part 2, Record Type Relationships

Create Record Relationships

In this exercise, you will add relationships to the AX Vehicle record. Record relationships allow easy access to data from related records. Once you build relationships into a record, you will be able to access the fields from related records without building queries. For this exercise, you will add four relationships to the AX Vehicle record: Maintenance, Category, Status, and Condition. You will reuse the existing Maintenance, Category, Status, and Condition records from the Acme Automobile Reference Application. Follow the steps below to add these record relationships.

1. In the **AX Vehicle** record, select the **Data Model** tab, and click **Add Relationship**.
2. Type **Maintenance** into the **Related Record Type** field, then click **Next**.
3. Set the **Relationship Name** as **maintenance**, and select **One to Many** as the **Relationship Type**. We use a One to Many type in this instance because one vehicle record can be related to multiple maintenance records.
4. Select **vehicleId** for both the **AX Vehicle** and **Maintenance** fields:



5. Click **Add**, and **Save Changes**.
6. Next, you will add a second relationship to the **AX Vehicle** record.
    - Click **Add Relationship**.
    - Type **Category** into the **Related Record Type** field.
    - Set the **Relationship Name** as **category**.

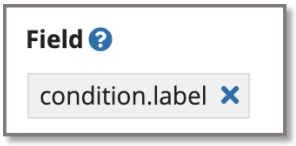
- Select **Many to One** as the **Relationship Type**.
- Select **vehicleCategory** for the **AX Vehicle** field and **id** for the **Category** field.
- Click **Add**, then **Save Changes**.

7. In the same manner, add two more relationships to the **AX Vehicle** record: **Status** and **Condition**:

    - Select **Many to One** for both relationship types.
    - Select **vehicleStatus** for the **Status** relationship and **vehicleCondition** for the **Condition** relationship.

8. Appian automatically creates user filters for Many to One record relationships. Click the **User Filters** tab to view the three newly created user filters. Make the following adjustments to the Condition, Status, and Category user filters:

    - Click the **Expression Editor** icon next to the filter name.
    - Scroll down to the **Field** field.
    - Delete the existing field, and replace it with **label**. For example, for the Condition filter, select **condition.label**:

    - Repeat these steps for all three user filters (Condition, Status, and Category).
    - Click **Save Changes**.

9. Click the link  next to the AX Vehicle record name to view and test these filters in the business user environment.

Use Record Relationships in Custom Record Fields

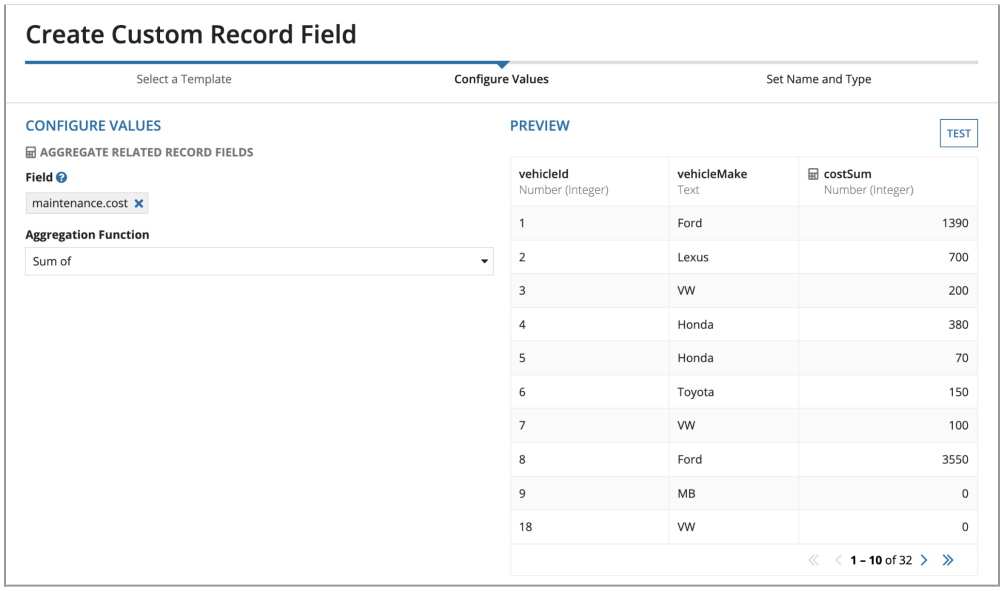
Record relationships can be very useful when you want to aggregate and display data from a related record. In this exercise, you will learn how to aggregate maintenance-related data using custom record fields. You will first create a custom record field for displaying the total cost of all maintenance per vehicle. Then, you will create a custom record field for showing the total count of all maintenance events per vehicle. Follow the steps below to create the two custom record fields.

1. In the **AX Vehicle** record under the **Data Model** tab, click **New Custom Record Field**.

2. Select **Aggregate Related Record Fields**. Click **Next**.

3. Select **maintenance.cost** in the **Field** dropdown menu.

4. Choose **Sum of** in the **Aggregation Function** dropdown menu:



5. Click **Test** to preview the new field, and click **Next**.

6. Leave the name as **costSum**, and click **Create**.

7. In the same manner, create a custom record field that displays a count of all maintenance requests for a vehicle.

    - Select the **maintenance.requestId** field.
    - Select the **Count of** aggregation function.
    - Rename the custom record field to **maintenanceRequestCount**.

8. Click **Save Changes**.

Update Summary View Interface

In this exercise, you will add a read-only grid with the vehicle's maintenance data to the AX_VehicleSummary interface. Also, you will update the Category, Status, and Condition fields, so that they display labels instead of numeric IDs. This can be done using the record relationships that you built in the previous exercise. Follow the steps below to complete this exercise.

1. Open **AX_VehicleSummary**, and place a **Read-Only Grid** component under the **Card Layout** with the vehicle details.

2. Select the **Read-Only Grid**, and make the following changes in the **Component Configuration** pane:

- Select **Record Type** as the **Data Source**, and search for the **Maintenance** record.
- Click **Filter Records**. In order to display the maintenance requests for the selected vehicle, you want to filter the record based on **vehicleId**.
- Click **Add Filter**. Select the **vehicleId** Field and the **=** Condition.
- Click the arrow **dropdown** next to **123**, and select **Expression**.
- Click the **null link** to edit the expression. In the **Expression Editor**, enter the following expression:

```
ri!vehicle[recordType!AX Vehicle.fields.vehicleId]
```

- Click **OK**. You will see that the read-only grid now only displays the maintenance requests that are associated with the record's vehicle ID.

3. Scroll down to the bottom of the **Component Configuration** pane, and click **Layout**. Change the **Label Name** to **Maintenance Requests**.

4. Click **Save Changes**.

Update Category, Status, and Condition Fields

Now that you have record relationships, let's update the Category, Status, and Condition fields in the AX_VehicleSummary interface. Currently, these fields display numeric IDs. You will replace them with more user-friendly labels. Complete the steps below.

1. Click the **Category** text box.

2. Click the **Display Value** link, and clear the expression.

3. Type the following expression:

```
ri!vehicle[recordType!AX
Vehicle.relationships.category.fields.label]
```

4. Click **OK**. The field may appear blank, but it will display the correct information once you view the interface in summary view.

5. In the same manner, update the **Status** and **Condition** fields. When selecting the relationship, select the **Status** record for the Status text box and the **Condition** record for the Condition text box.

6. Click **Save Changes**. Return to the AX Vehicle record, and click the link next to the AX Vehicle record name. Click on a **VIN** to view the updated summary view interface.

Add an Expression-Based User Filter

In Exercise 5, you created a simple user filter to use in your AX Vehicle record. In this exercise, you will create an expression-based user filter to filter the AX Vehicle record by vehicle make. Follow the steps below to create an expression-based user filter.

1. In the **AX Vehicle** record, click **User Filters**.
2. Click **New User Filter**, and select the **Expression** radio button.
3. Name the filter **Make**, and type the following expression into the **Expression Editor.** As a reminder, this expression should be typed. If you copy and paste this expression into the editor, you will need to edit the lines with recordType.

```
a!localVariables(
local!vehicleMakes: a!queryRecordType(
  recordType: recordType!AX Vehicle,
  fields: a!aggregationFields(
    groupings: {
      a!grouping(
        Field: recordType!AX Vehicle.vehicleMake,
        alias: "make"
      )
    },
    measures: {
      a!measure(
        field: recordType!AX Vehicle.vehicleMake,
        function: "COUNT",
        alias: "count"
      )
    }
  ),
  pagingInfo: a!pagingInfo(
    startIndex: 1,

          batchSize: 5000
        )
     ),
     a!recordFilterList(
       name: "Make",
       options: a!forEach(
         items: local!vehicleMakes.data,
         expression: a!recordFilterListOption(
           id: fv!index,
           name: fv!item.make,
```

```
            filter: a!queryFilter(
              field: recordType!AX Vehicle.vehicleMake,
              operator: "=",
              value: fv!item.make
            ),
            dataCount: fv!item.count
          )
        )
      )
    )
```

This expression first creates a local variable for the record query, so that the record query can be easily referenced in the second half of the expression. In this case, a!queryRecordType() executes a query on the AX Vehicle record type and returns the vehicle makes as the result. Instead of manually creating an option for each unique make, the a!forEach() function writes these options for you based on vehicle makes returned in the record query. The a!forEach() function will take the array of vehicle makes and pass them to an expression one at a time, creating the a!recordFilterListOption function for each item in the array.

4. Click **OK**, and then **Save Changes**. Click the link  located next to the record name to access the AX Vehicle record. You will see the Make filter available at the top of your record list. Test that it works as expected.

TIP: Use Local Variables to Define and Store Temporary Values

---

- A local variable is a type of variable that is used to define and store temporary values within an expression. They are useful when you only need values within a particular expression. For example, if you need to temporarily store user-inputted search terms in an interface, this can be easily accomplished by adding a local variable local!search to the interface.
- A local variable will temporarily hold a value, until you save it. For example, in an interface you can store a local variable using a related rule input.
- Use the following syntax to define a local variable: a!localVariables(localVar1, localVarN, expression). LocalVarN means that you can have multiple local variables.
- Local variables are frequently used in interfaces. Although you have to add them in Expression Mode, the advantage is that you can declare as many local variables as you need.

Appian
Academy

Appian Step-by-Step #7

Exercise to Accompany
Query Entities

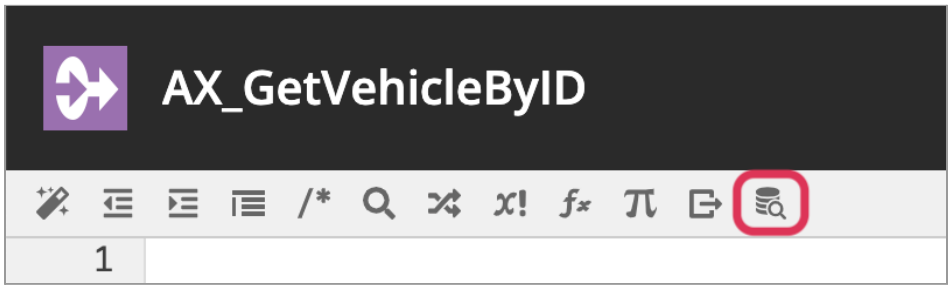
The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 7: Query Entities

Create a Query

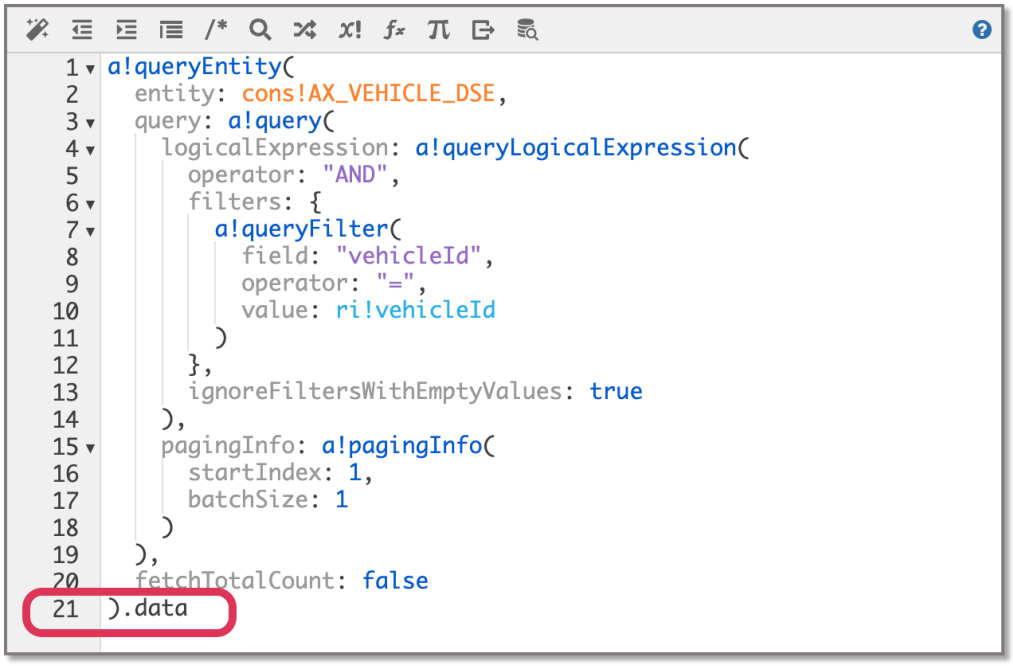
In this exercise, you will create an expression that will return information about a specific vehicle in the fleet given its ID. You will use the Query Editor to create this query. This query will be used in a later exercise to help you configure the Request Maintenance action.

1. In your application, click **New**, and select **Expression Rule** from the dropdown menu.
2. Name the rule **AX_GetVehicleByID**, and give it a description: "Return a vehicle by its unique ID number." Save the rule in **AX Expressions**, and click **Create**.
3. Click the **Query Editor** icon to launch the Query Editor:



4. Enter **AX_VEHICLE_DSE** in the **Data Store Entity** field. This constant was auto-generated for you when you created a record action. Click **Continue**.
5. Click the **Rule Inputs** button in the top right corner.
6. Click **Add New Rule Input.** Name this input **vehicleId**, set the **Type** to **Number (Integer)**, and click **Save**.
7. Under **Paging & Sorting**, replace 50 with **1** row per page.
8. Under **Filters**, click **Add Filter**, then set the following configurations:
    - Under **Field**, select **vehicleId**.
    - Under **Value**, use the **small arrow** next to 123 to select **Input/Variable.**
    - Under **Value**, use the dropdown menu to select **ri!vehicleId.**
9. Click **GENERATE QUERY**, and then **Save Changes**.
10. To test this query, type any vehicle ID into the Value field. For example, type **1**, and click **Test Rule**. You will see the entire data subset in the **Test Output**.

11. Add **.data** to the end of your expression to limit the query results only to the vehicle data:



```
a!queryEntity(
  entity: cons!AX_VEHICLE_DSE,
  query: a!query(
    logicalExpression: a!queryLogicalExpression(
      operator: "AND",
      filters: {
        a!queryFilter(
          field: "vehicleId",
          operator: "=",
          value: ri!vehicleId
        )
      },
      ignoreFiltersWithEmptyValues: true
    ),
    pagingInfo: a!pagingInfo(
      startIndex: 1,
      batchSize: 1
    )
  ),
  fetchTotalCount: false
).data
```

12. Click **Test Output** again to view the impact of the .data notation on the returned result. Click **Save Changes**.

Appian
Academy

Appian Step-by-Step #8

Exercise to Accompany
Process Modeling 101: Automate Your Business Processes

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 8: Process Modeling 101, Part 1

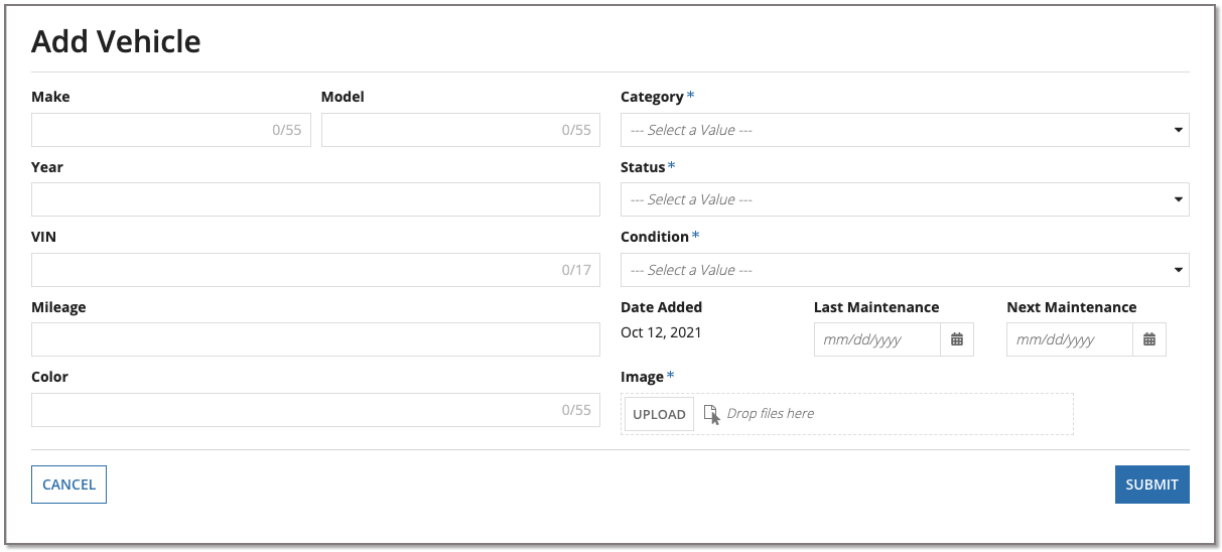
In this exercise, you will edit a process model for adding new vehicles that was generated for you, along with other objects, when you created a record action. You will edit both the process model and related interface to ensure that they function correctly and meet application requirements. These requirements are:

- The process allows the Registrar to fill out the Add New Vehicle form
- The process allows the Supervisor to approve the vehicle
- If the vehicle is approved, the process writes all vehicle details to the database

Before you proceed to extend your auto-generated process model, create the interfaces that this process model will need.

Edit the Add Vehicle Form

In this section, you will edit the auto-generated AX Add Vehicle interface that the AX New Vehicle process model uses to kick off the addition of the vehicle to the fleet. Your goal is to improve the overall UX of this form and to update a couple of fields, so that they display the information correctly. After edits, this interface will look like the image below.



Complete the steps below to edit the Add Vehicle form.

Update Fields and Labels

1. Open **AX_AddVehicle**, and update the form title to **Add Vehicle**.
2. Click the **record** rule input in the top right corner, and rename it to **vehicle**.
3. Delete the following fields: Vehicle Status, Vehicle Added By, Vehicle Last Modified By, Vehicle Condition, Vehicle Category, Vehicle Image, and Vehicle Last Modified Date.

4. Remove redundant labels. All default labels in this form contain **"vehicle,"** which is unnecessary in the Add Vehicle form. Also, remove the word **"date"** from the labels for the Last Maintenance and Next Maintenance fields.

5. Drag and drop a **side-by-side** layout at the top of column one. Then drop the **Make** and **Model** fields inside of the side-by-side layout. The side-by-side layout will ensure that the related Make and Model fields stay next to each other on a narrower screen.

   Keep in mind that columns layouts are flattened into a single column on mobile devices. If you need certain related fields to stay together, use side-by-side layouts.

6. Add another **side-by-side** layout in column two, and place the **Date Added**, **Last Maintenance**, and **Next Maintenance** inside of it. For reference, use the image at the top of this section.

7. Next, add a validation to the **Next Maintenance** field. Let's create a validation so that the Next Maintenance date has to be after the Last Maintenance date. Click **Next Maintenance**, and scroll down to **Validations** in the **Component Configuration** pane.

8. Click **List of Text** or the **Expression Editor** icon. Click **Add Item**.

9. In the **Expression Editor**, type the following expression:

   ```
   if(ri!vehicle.vehicleNextMaintenanceDate <
   ri!vehicle.vehicleLastMaintenanceDate, "The next maintenance
   date must be after the last maintenance date.", null)
   ```

   This expression checks to see if the next maintenance date is before the last maintenance date. If it is, the field displays an error message. If not, the field is valid and displays no error message.

10. Click **OK**, then **Save Changes**.

Add a File Upload Component

Next, add the Image field back as a File Upload component.

1. Drag and drop a **File Upload** component onto the canvas. Edit the field label to **Image**.

2. In the **Component Configuration** pane, scroll to the **Target Folder** field**.** Type and select the constant **AX_DOCUMENTS_FOLDER_POINTER**.

3. For the **Selected Files** and **Save Files To** fields, select **ri!vehicle.vehicleImage** using the dropdown menu.

4. Scroll down, and select the **Required** checkbox.

Add Dropdown Components

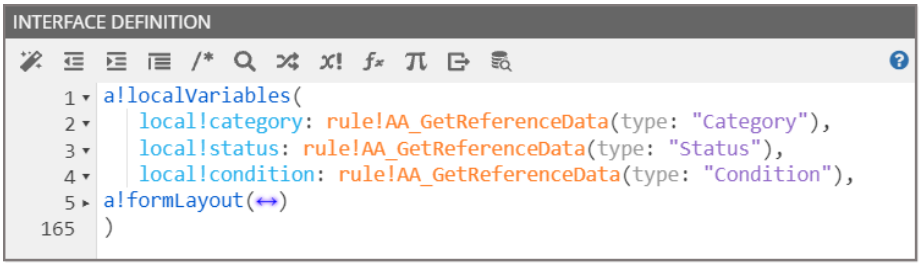
In this section, you will add the Category, Status, and Condition fields back as Dropdown components. You will then populate these dropdowns using data stored in a reference database table. For the sake of exercise, you will draw on data and objects that already exist in the Acme Automobile Reference app. Complete these steps to add dropdown components.

1. Drag and drop three **Dropdown** components onto the canvas. Edit their labels to **Category**, **Status**, and **Condition**.

2. To draw on the values stored in a reference table, you will need to create three local variables. Click **Expression Mode** </> EXPRESSION MODE to access the expression for your interface.

3. Click the little arrow in line 1 to collapse the interface expression, and then press **Enter**. Type and select **a!localVariables**, and then cut the **closing parenthesis**, and paste it at the end of the expression. Press **Enter** again, and type the following expression in lines 2 through 4:

```
local!category: rule!AA_GetReferenceData(type: "Category"),
local!status: rule!AA_GetReferenceData(type: "Status"),
local!condition: rule!AA_GetReferenceData(type:
"Condition"),
```



You now have three local variables that you will use to define the values in the Category, Status, and Condition fields. Click **Save Changes**.

4. Switch to the design mode by clicking the **Design Mode** button DESIGN MODE, and save changes.

5. Next, configure the Category field to display correct choices in the dropdown menu. Make sure the Category field is selected, and scroll to **Choice Labels** in the **Component Configuration** pane. Click the **Expression Editor icon** next to it, and enter the following expression:

```
local!category[recordType!Category.fields.label]
```

6. Scroll to **Choice Value**s, and click the **Expression Editor** icon next to it. Enter the following expression:

```
local!category[recordType!Category.fields.id]
```

7. Scroll to **Selected Value** and **Save Selection To**. Use the in-line arrow to select **ri!vehicle.vehicleCategory** in both fields.

8. In the same way, configure the **Status** and **Condition** fields.

    - For the **Status** field, use `local!status[recordType!Status.fields.label]` in **Choice Labels**, and `local!status[recordType!Status.fields.id]` in **Choice Values**.
    - For the **Condition** field, use `local!condition[recordType!Condition.fields.label]` in **Choice Labels** and `local!condition[recordType!Condition.fields.id]` in **Choice Values**.

9. Configure the **Selected Value** and **Save Selection To** fields for Status and Condition. Make sure to select the corresponding fields for Status and Condition.

10. Scroll down and select the **Required** checkbox for all three dropdowns.

11. The dropdown fields should now display the correct choices. Review, and click **Save Changes** to save your work.

TIP: Use the Visibility Setting to Conditionally Show Interface Components

---

- In some cases, you may want to only display a component based on a condition.
- To do this, navigate to the **Visibility** setting in the **Component Configuration** pane, then select **Only Show When**. Use the **Expression Editor** to write an expression.
- For example, imagine that you want to display a Comments box if a vehicle's mileage is above 150,000 miles. You can do this by adding the expression **ri!vehicle.vehicleMileage > 150000**. This expression determines whether the Comments component is displayed on the interface or not. When set to false, the component is hidden and not evaluated. The default is set to true.

Configure the Submit Button

In this step, you will update the Date Added value to display the current date. While the Registrars won't fill out the vehicle added date using the Add Vehicle form, this important date still needs to be recorded to the database table for each new vehicle in the fleet. Since it is going to be a read-only field and not a user input field, you will need to configure the Submit button to auto-save the Date Added value.

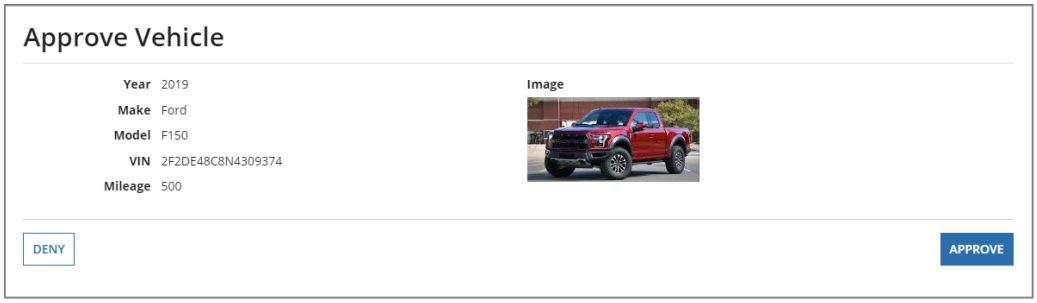
1. Click the **Date Added** field.
2. In the **Component Configuration** pane, scroll down to **Display Value**, and click the **Expression Editor**.
3. Clear the existing expression, and type **today()**. Click **OK**.
4. Scroll down, and select the **Read-only** checkbox.
5. Click the **Submit** button.
6. Click the **Expression Editor** for **Save Value To**, and type the following expression:

   ```
   a!save(ri!vehicle.vehicleDateAdded, today())
   ```

   In this expression, **ri!vehicle.vehicleDateAdded** is the target and **today()** is the value being saved into the target.
7. Click **OK**, then **Save Changes**.

Create the Supervisor Approval Form

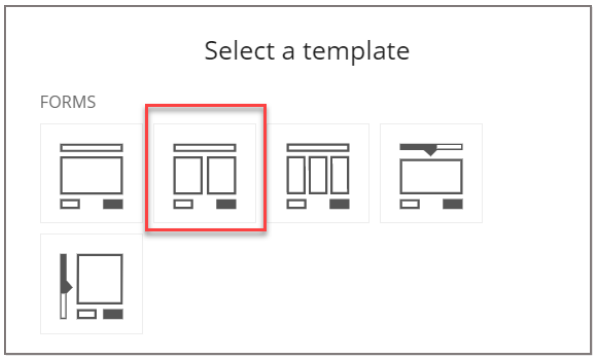
In this part, you will create an interface for the Supervisor to approve new vehicles. After the Registrar submits a new vehicle, the Supervisor will receive a task to approve or reject the vehicle. The interface that you create in this exercise will look like the image below:



Follow the steps below to create a Supervisor approval form.

1. In your application, click **New** and select **Interface**. Name it **AX_SupervisorForm**, and add a simple description: "Displays read-only details about a selected vehicle for Supervisor approval."
2. Save the interface in **AX Interfaces**, and click **Create**.

3. Choose the **Two-Column Form** template:



4. Click the **plus** sign in the Rule Inputs, and add the new rule input **vehicle**. Select the CDT **AX_Vehicle** in the **Type** field, and click **Create**.

5. Rename the **cancel** rule input to **approvalDecision**, but leave **Boolean** as the type.

6. Rename the form to **Approve Vehicle**, and delete the top **Section Layout**. To delete, select this section, and use the dropdown arrow to navigate to **Delete**.

7. Delete the **blue Section label** for the bottom section layout.

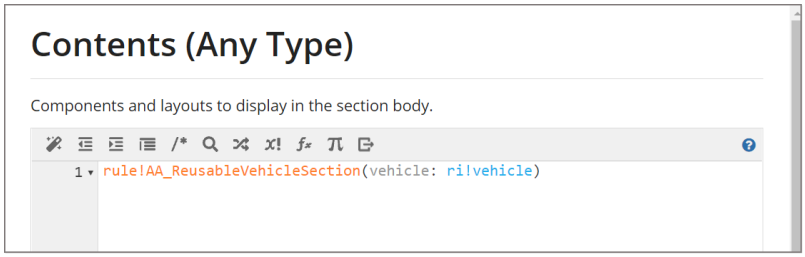
8. Drag and drop **three text** and **two integer components** into the first column.

9. Change the labels for the text fields to **Make**, **Model**, **VIN**, and for the integer fields to **Year** and **Mileage**. For each of these fields, make the following changes in the **Components Configuration** pane:

    - Select **Adjacent** in the **Label Position** dropdown.
    - Select the correct values for **Display Value**. Use the dropdown menus to select the values. For example, for Vehicle Year, select **ri!vehicle.vehicleYear** from the dropdown menu.
    - Check the **Read-only** checkbox.

TIP: Save Time by Creating Reusable Interfaces

- As you plan your first application, keep in mind that interfaces can be built as reusable blocks. Simply create a reusable interface, and then call it from other interfaces in your app. For example, the read-only vehicle details that you just added to the supervisor approval form could have been set up as its own interface. To call a reusable interface from a different interface, use **rule!** and the name of the interface you want to reuse:



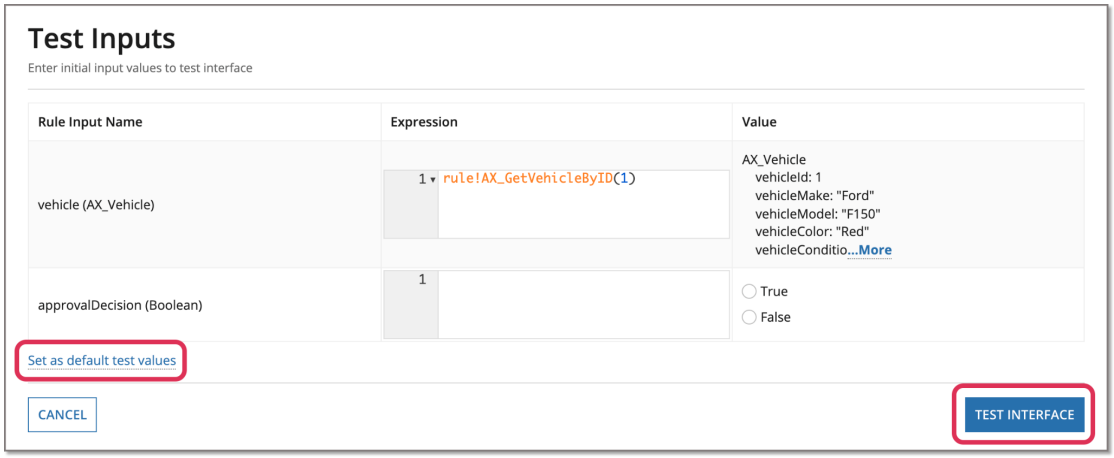
- This is a particularly useful technique for reporting interfaces. You can create each chart or grid as its own interface, and then combine them into different larger reporting interfaces that you may need for the business users of your app.

---

10. Next, drag and drop an **Image** component into the second column. Delete the label **Image**, and click the **+** in the Image field. Select **Document Image**. In the **Component Configuration** pane, click **a!EXAMPLE_DOCUMENT_IMAGE()**, and delete all text in the **Expression Editor**. Type the following expression:

```
ri!vehicle.vehicleImage
```

11. You will see an error message. This error appears because the value for the document cannot be null. Let's fix this by adding test values to this interface. Click **Test**, and type the following expression into the vehicle row: `rule!AX_GetVehicleByID(1)`. Click the **Set as default test values** link, and then click **Test Interface**.



You will see the error disappear, and the interface will display the sample data.

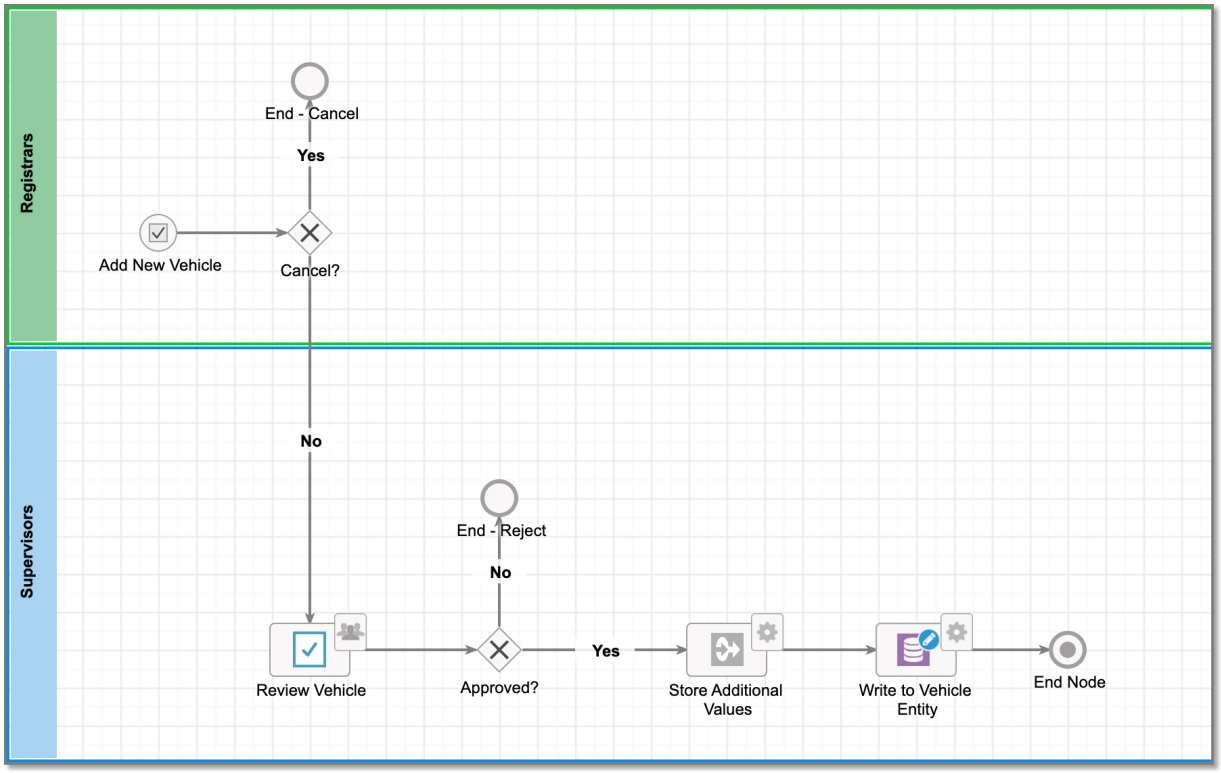
12. Select the **Cancel** button, and rename it to **Reject**. Click the **link** under **Value**, and change it to **false**. Select **ri!approvalDecision** in the **Save Value To** field.

13. Select the **Submit** button, and rename it to **Approve**. Click the **link** under **Value**, and make sure it is set to **true**. Select **ri!approvalDecision** in the **Save Value To** field. Click **Save Changes**.

Edit the Add Vehicle Process Model

The Add Vehicle record action has already created a preliminary process model for you. In this exercise, you will edit this auto-generated process model to include the steps for the Supervisor approval of the vehicle addition to the fleet. Keep in mind that because this process model was auto-generated, security is already set for this model. In other instances, you will need to secure each new process model individually.

At the end of this exercise, your process model will look like the image below.

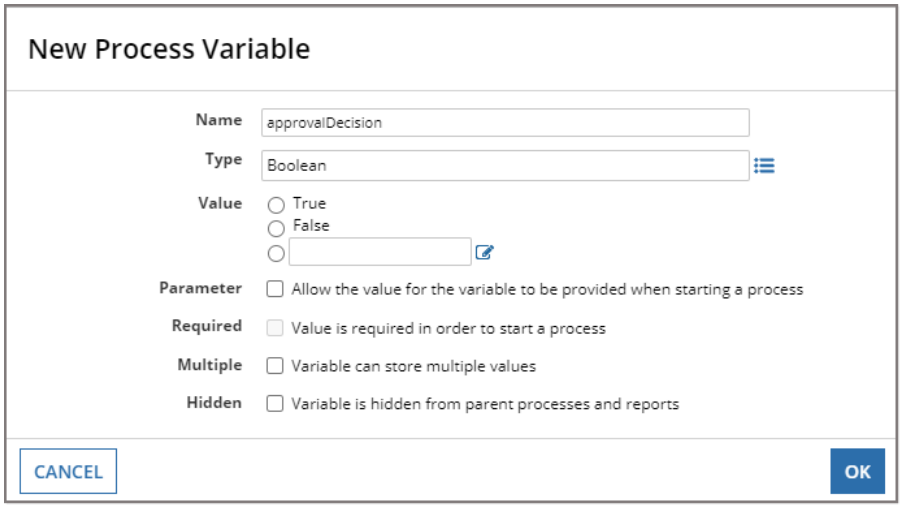


Follow the steps below to complete this exercise.

Configure Process Properties

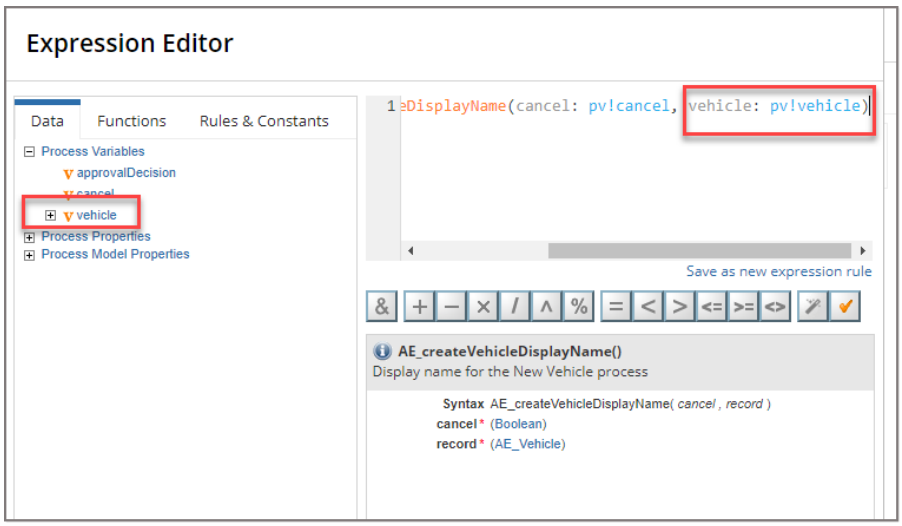
Let's start by configuring the process model properties, such as variables, alerts, and data management. Because this process model was auto-generated, some of the properties are already configured. Follow the steps below to review and edit process properties:

1. Open the **AX New Vehicle** process model.
2. If you are in the Analyst View, switch to **Designer View**. Simply click the button in the upper right corner of the process modeler.
3. Click the **Properties** button  in the toolbar.
4. Navigate to the **Variables** tab. You need to add a new process variable **approvalDecision** that you will use to configure the Approved? XOR node later. Click **Add Variable**, and name the new variable **approvalDecision**. Type **Boolean** into the **Type** field, and click **OK**.



5. Click the **record** variable, and update the name to **vehicle**.
6. In the **General** tab, click the **Expression Editor icon** next to the **Process Display Name** field. The AX_createVehicleDisplayName expression was created alongside the record action. This expression rule creates a dynamic display name that users can see after a process has been started.
7. Before you use this expression rule, open the rule in a new tab. This can be done by holding the ctrl/command key and clicking the name of the rule within the Expression Editor. Once the expression rule is open, change the name of the **record** rule input to **vehicle**. Click **Save Changes**.

8. Click in the expression, and delete **record: pv!record**. Type **vehicle:**, and insert **pv!vehicle** by selecting it in the **Data** tab.



Click **Save and Close.**

9. Navigate to the **Process Start Form** tab to review the interface that starts the process. Make sure the rule inputs for this interface are **vehicle** and **cancel**.

10. Navigate to the **Alerts** tab to review the alert settings. Make sure that the **AX Administrators** are selected to receive alert notifications. Update the group if necessary.

11. Navigate to the **Data Management** tab to adjust the timeline for process archiving. The system default is to archive processes 7 days after completion. Change this to **3** days to improve utilization of server resources. This setting will archive process history, metrics, and variables from memory after three days.

12. Click **OK**, then **File > Save**.

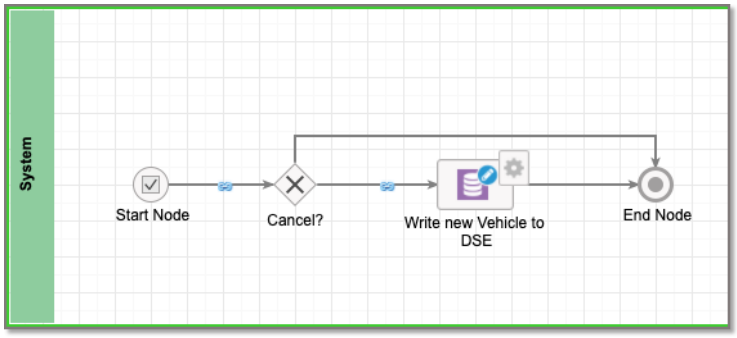
TIP: Use the Analyst View to Draft High Level Diagrams

Note that you can select to view a process model using the Analyst or Designer views. The process Analyst View allows you to draft high-level diagrams, while the process Designer View allows you to configure the nodes and publish process models. Use the Analyst View when drafting a process, for example if you want to share a process diagram with stakeholders or developers. This will suppress any errors and validation messages.

You can select the views from the upper right corner in the Process Modeler:

Configure the Cancel XOR Gateway

Gateways are added to the process when you need to branch the workflow. In this process, if the Registrar cancels the start form, the process will end. If the Registrar submits the form, the process will continue to the next process node. Appian has created a template for you that you will need to edit before it becomes fully functional.



Follow the steps below to configure the Cancel? Gateway.

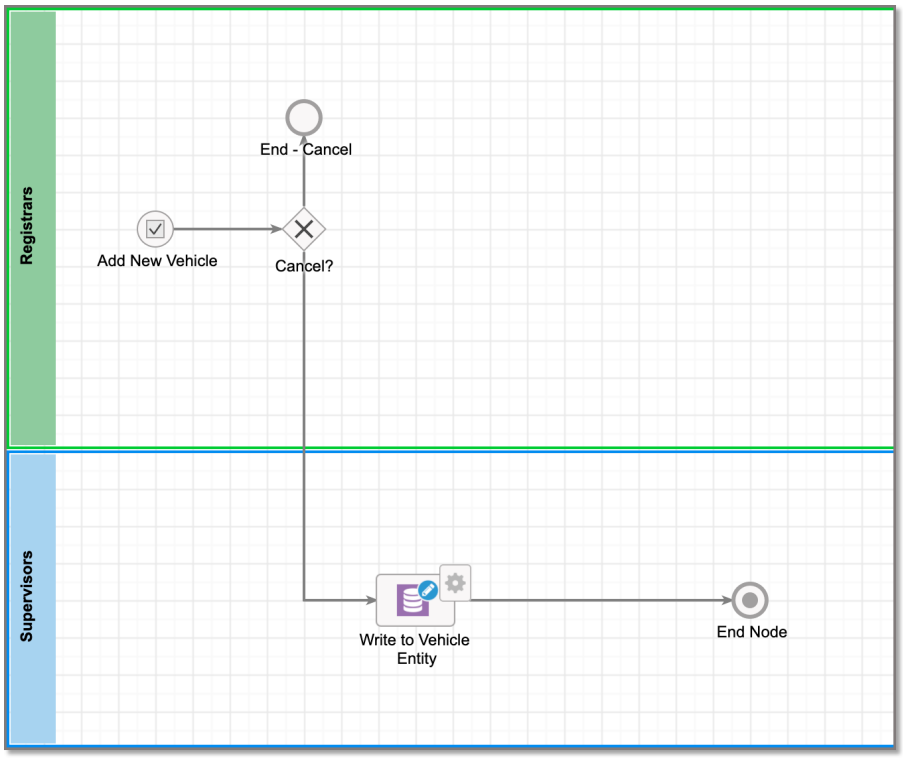
1. Switch to **Designer View** in the top right corner.
2. Click **System** in the first swim lane, and rename it to **Registrars**. Add a swimlane for **Supervisors**. Click the **Add Horizontal Lane** button  in the toolbar to add one swimlane, and rename it to **Supervisors**.
3. Rename the **Start Node** to **Add New Vehicle**.
4. Drag a new **End Event** above the Cancel? XOR gateway in the Registrars swimlane. Rename it to **End - Cancel**.
5. Delete the **line** connecting the **Cancel? XOR gateway** and the **End Node**.
6. Connect the **Cancel? XOR gateway** and **End - Cancel** nodes. To connect, hold down **SHIFT** on your keyboard, and then click from one node to another. SHIFT will automatically toggle your pointer to the connector tool. You can also use the **Connect** icon  to connect the nodes.

7. Double-click **Cancel?**, and navigate to the **Decision** tab.
8. For **pv!cancel**, select the **End - Cancel** in the **Result** column.
9. For the **Else if none are TRUE** condition, select **Write new Vehicle to DSE** in the **Result** column, and click **OK**.
10. Click **File > Save**.

Configure the Write to Vehicle Entity

Before you test your preliminary process model, configure the Write to Data Store Entity smart service to ensure that all vehicle data is written to the database table. Follow the steps below to complete this task.

1. Rename the **Write new Vehicle to DSE** node to **Write to Vehicle Entity**.
2. Delete the **line** connecting **Cancel?** and **Write to Vehicle Entity**.
3. Move the **Write to Vehicle Entity** and **End Node** nodes to the Supervisors lane.
4. Connect the **Cancel?** And **Write to Vehicle Entity** nodes.
5. Double-click **Cancel?**, and navigate to the **Decision** tab. Make sure **Write to Vehicle Entity** is selected for the **Else if none are TRUE** condition in the **Result** column. Click **OK**:



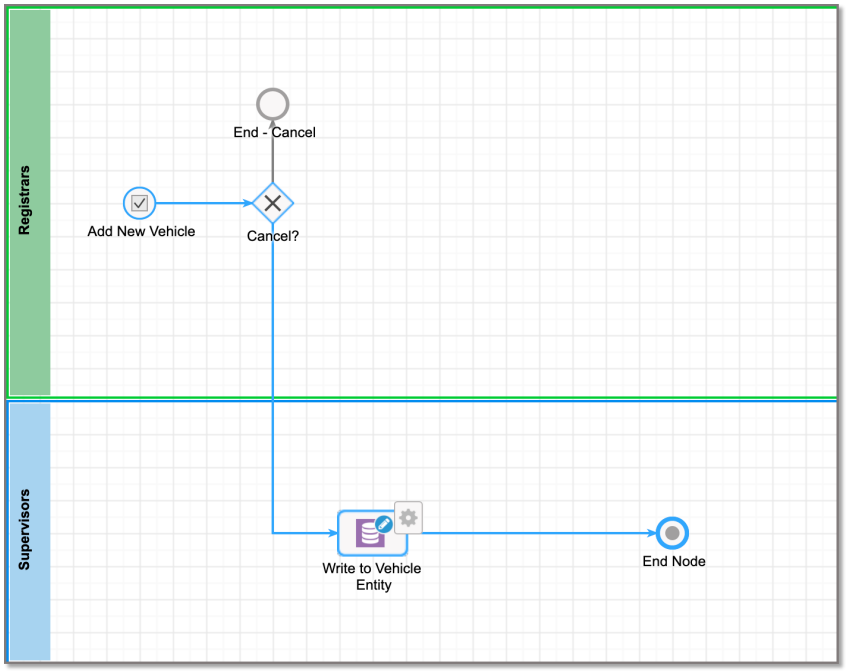
6. Double-click **Write to Vehicle Entity**.
7. Navigate to the **Data** tab, and then to the **Inputs** tab.
8. You will notice that two node inputs have been already created for you. This was done when you created the Add Vehicle record action. Click **record**, and update the name to **vehicle**.
9. Select **vehicle** from the **Value** dropdown field. It will display as pv!vehicle when selected.
10. Navigate to the **Assignment** tab. Use this tab to assign the Write to Vehicle Entity activity to process designers. This is done to elevate the security of this node since writing to the application's Data Store Entity is not something that should be done by basic users, and it typically requires a higher level of permission. Select the **Run as whoever designed this process model** radio button, and click **OK**.
11. Click **File** > **Save and Publish**.

Test the Process

Testing should be performed each time you add a new piece of functionality to the process model. Be sure to test frequently to make sure your process model works as expected. Follow the steps below to test the process.

1. Click **File**, and select **Start Process for Debugging**.
2. Complete the **Add Vehicle** form, and click **Submit**. Since these are test values, feel free to make up vehicle details to enter into the form.
3. Once you submit, you will notice that a new process instance has started in the **Monitoring Mode** and on a new tab in the Process Modeler.

4. Click the **Refresh** button  in the toolbar to update the process flow. You will notice that the process will advance to the **End Node** node:



5. Click the **Process Details** button  in the toolbar, and navigate to the **Variables** tab to verify that the data you entered into the form was captured by the process model.

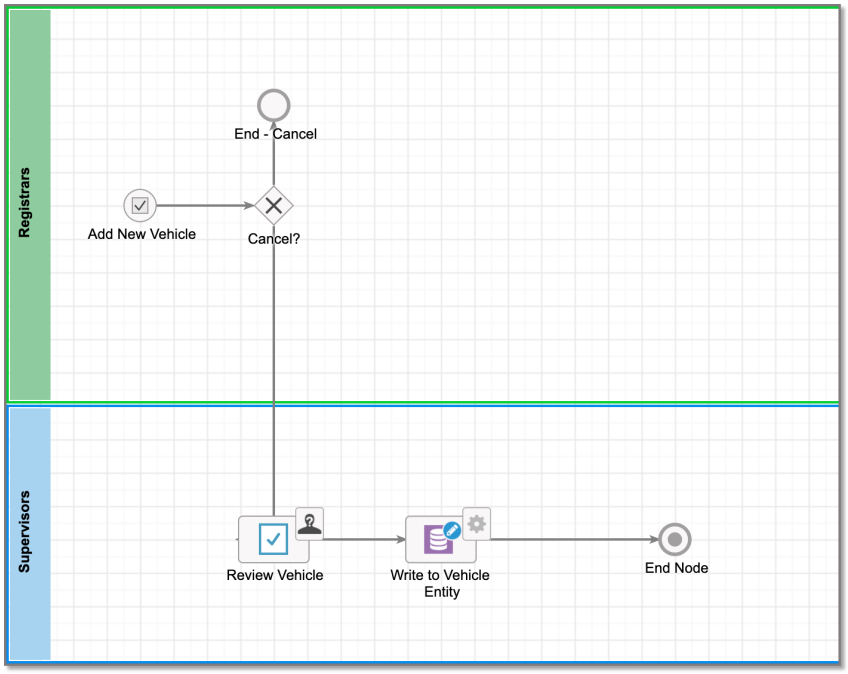
6. Exit out of the **Monitoring** tab to return to your process model.

Configure the User Input Task

Next, let's configure the Review Vehicle node. You will use this node to add the Supervisor approval form to this process. Complete the steps below to add and configure a user input task.

1. Drag a new **User Input Task** node to the Supervisor swimlane to the left of Write to Vehicle Entity. Rename it to **Review Vehicle**.

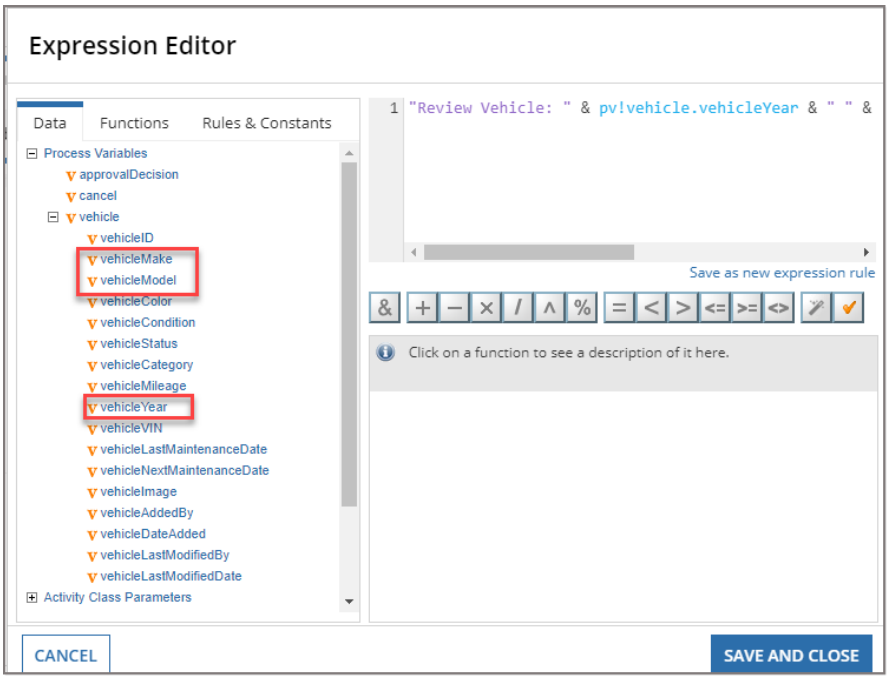
2. Reconnect the nodes so that **Review Vehicle** falls in between **Cancel?** and **Write to Vehicle Entity**:



3. Double-click **Cancel?**, and navigate to the **Decision** tab. Make sure **Review Vehicle** is now selected for the **Else if none are TRUE** condition in the **Result** column. Click **OK**.

4. Double-click **Review Vehicle**. First, edit the **Task Display Name** to make it dynamic. Click the **Expression Editor icon**, and select the process variables for the **year**, **make**, and **model** to insert them into the expression. Your expression should look as follows:

```
"Review Vehicle: " & pv!vehicle.vehicleYear & " " &
pv!vehicle.vehicleMake & " " & pv!vehicle.vehicleModel
```

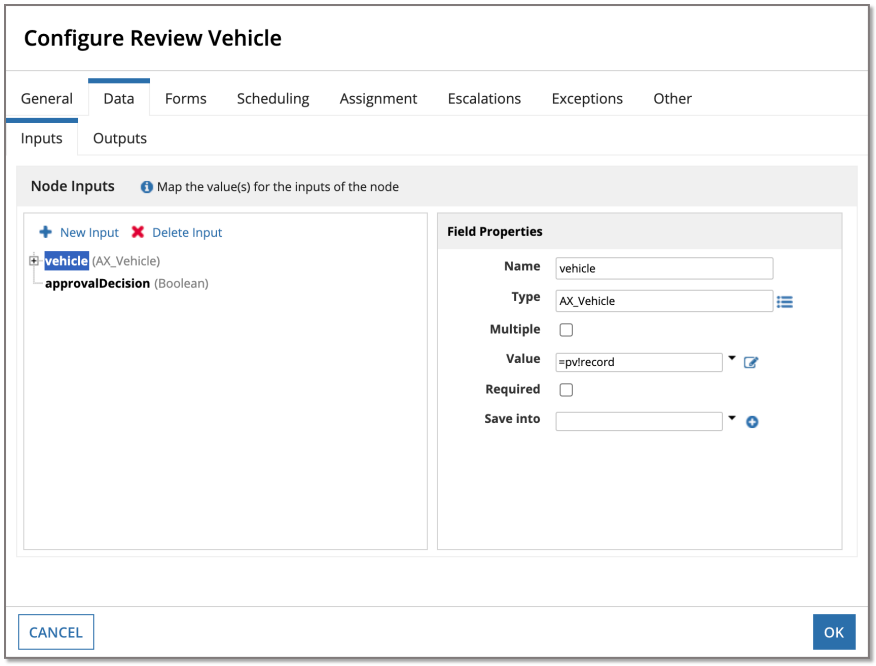


5. Navigate to the **Forms** tab, and use the **Directory** icon  to select the **AX_SupervisorForm**. Allow the process model to automatically create node inputs to match your interface's inputs.

6. Navigate to the **Assignment** tab. In the **Assign to the following** field, type and select **AX Supervisors** from the list of auto-suggestions.

7. Navigate to the **Data** tab. You will need to configure the inputs in this tab to ensure that all existing vehicle data flows into the form, and the new data, entered by the Supervisor, is captured as well.

8. Select the **vehicle** node input. Click the dropdown menu next to the **Value** field, and select **vehicle**. This will allow you to display the existing vehicle data in the form. You don't need to update the **Save into** field because the Supervisor will not modify any vehicle information.



9. Select the **approvalDecision** node input. Click the dropdown menu next to **Save into**, and select the **approvalDecision** process variable. If the Supervisor clicks **Deny**, this will save the approvalDecision value as false into the process.

10. Click **OK**, and then save the process model by clicking **File** > **Save & Publish**.

Test the Process

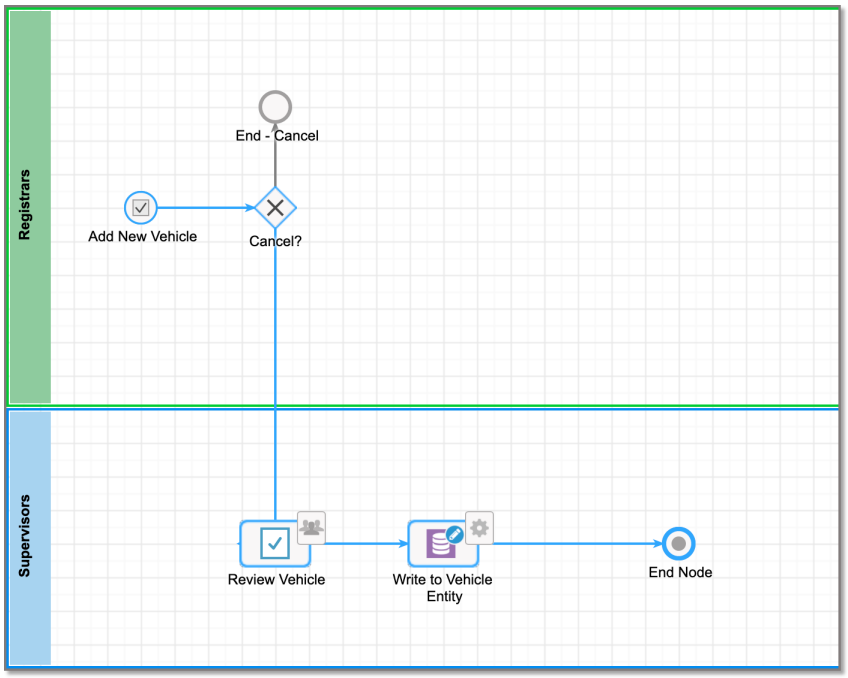
1. Click **File**, and select **Start Process for Debugging**.

2. Complete the **Add Vehicle** form, and click **Submit**. Since these are test values, feel free to make up vehicle details to enter into the form.

3. Click the **Refresh** button  in the toolbar to update the process flow. You will notice that the process will advance to the **Review Vehicle** node.

4. Right-click the **Review Vehicle** node, and select **View Form**. Click **Accept** to accept the task, and then click **Approve**.

5. Click the **Refresh** button  once again to update the process flow. You will notice that the process will advance to the **End Node** node:



6. Click the **Process Details** button  in the toolbar, and navigate to the **Variables** tab to verify that the data you entered into the form was captured by the process model.

7. Exit out of the **Monitoring** tab to return to your process model.

Configure the Approved? XOR Gateway

In this process, if the Supervisor denies the AX_SupervisorForm form, the process will end. If the Supervisor approves the form, the process will continue to the next process node. Follow the steps below to configure the Approved? XOR node.

1. Drag and drop a **XOR** node to the right of Review Vehicle. Rename it **Approved?**

2. Drag and drop an **End Event** node above Approved? Rename it **End - Reject**.

3. Connect the **Approved?** and **End - Reject** nodes.

4. Double-click **Approved?**, and navigate to the **Decision** tab.

5. Click **New Condition**, and then click the **Expression Editor** icon next to the first condition. Select the **approvalDecision** process variable. Click **Save and Close**.

6. In the **Results** column, in the first dropdown, select **Write to Vehicle Entity**.

7. In the **Results** column, in the second dropdown for the **Else if none are TRUE** condition, select **End - Reject**.

8. Click **OK**, and save the process model by going to **File > Save and Publish**.

Configure the Script Task

A script task is used to perform an automated activity. For this process, you will use a script task to store three variables: Vehicle Last Modified Date, Vehicle Last Modified By, and Vehicle Added By. In this example, the script task is used because we don't want the user to enter these values manually. Instead, we will lift the date and the name of the user from the process automatically. Follow the steps below to set up the script task.

1. Drag and drop a **Script Task** to the right of Approved? Rename it **Store Additional Values**.

2. Double-click **Approved?**, and navigate to the **Decision** tab.

3. Make sure **Store Additional Values** is selected in the first dropdown of the **Results** column. Click **OK**.

4. Double-click **Store Additional Values**, and navigate to the **Data** tab. Click the **Outputs** tab.

5. Click **New Custom Output**. You will want to set the last modified date as the current date. Click the **Expression Editor** icon, and type the function **today()**. Click **Save and Close**.

6. For the **Target**, click the **inline arrow,** and hold your cursor over **vehicle**. Select **vehicle.vehicleLastModifiedDate**.

7. Click **New Custom Output**. You will now want to set the last person to modify the record as the initiator of the process model. Click the **Expression Editor** icon, and type the function **pp!initiator**. Alternatively, click **initiator** under **Process Properties**:

8. Click **Save and Close**.
9. For the Target, select **vehicle.vehicleLastModifiedBy**.
10. Click **New Custom Output** once again. You will now want to set the person who added the vehicle. Click the **Expression Editor** icon, and type the function **pp!initiator**. Click **Save and Close**.
11. For the target, select **vehicle.vehicleAddedBy**.
12. Click **OK**, save the process model by clicking **File > Save and Publish**.

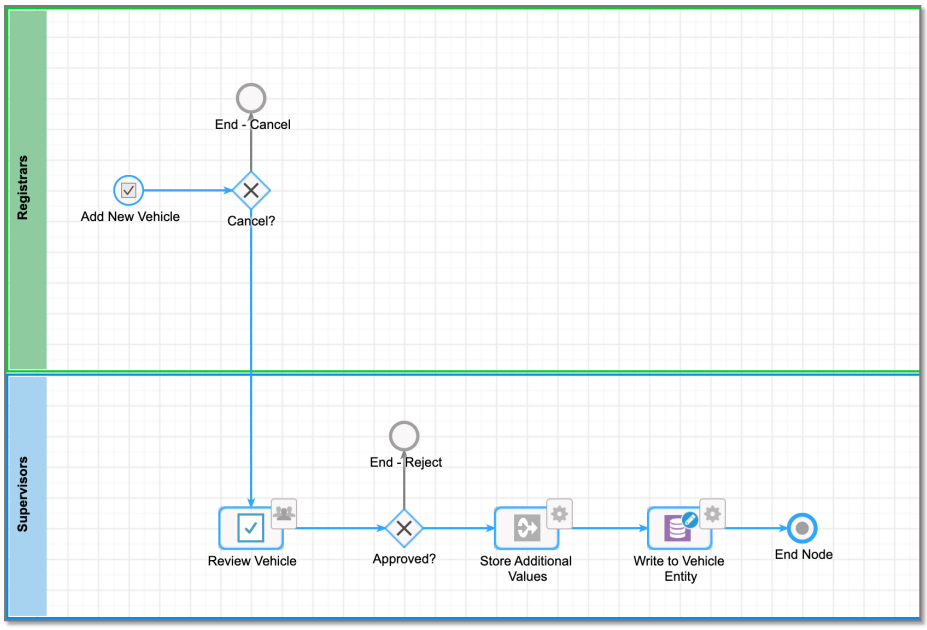
Test the Process

1. Click **File**, and select **Start Process for Debugging**.
2. Complete the **Add Vehicle** form, and click **Submit**. Since these are test values, feel free to make up vehicle details to enter into the form.

3. Click the **Refresh** button  in the toolbar to update the process flow. You will notice that the process will advance to the **Review Vehicle** node.

4. Right-click the **Review Vehicle** node, and select **View Form**. Click **Accept** to accept the task, and then click **Approve**.

5. Click the **Refresh** button  once again to update the process flow. You will notice that the process will advance to the **End Node** node:



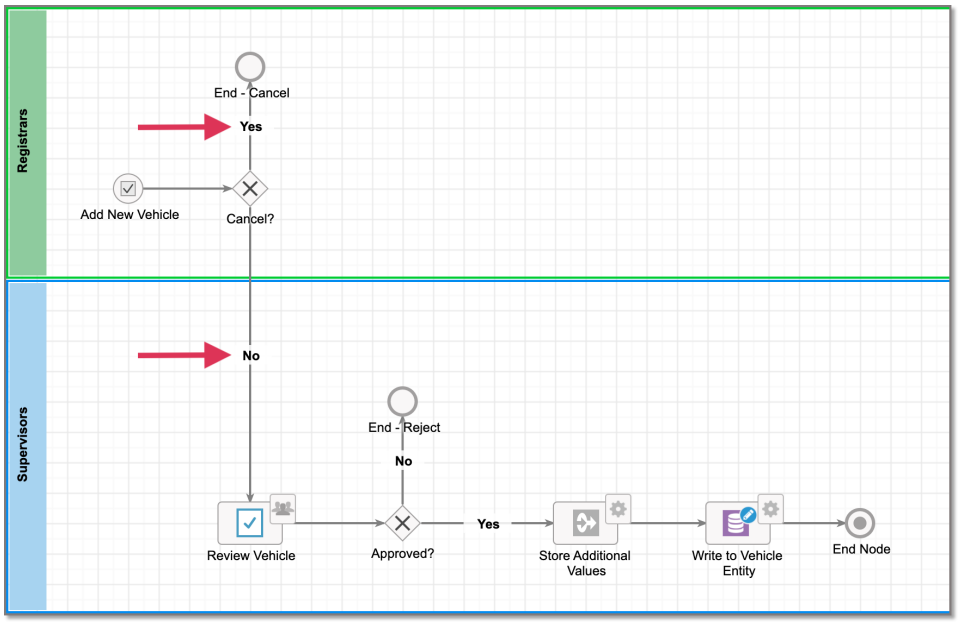
6. Click the **Process Details** button  in the toolbar, and navigate to the **Variables** tab to verify that the data you entered into the form was captured by the process model.

7. Close the process monitoring tab, but keep the process model open. Debug the process model once again, but this time for the **Reject** flow. Check that the process arrives at the **End - Reject** flow.

Add Flow Labels

Flow labels help keep your process models clear and organized. Follow the steps to add flow labels to your process:

1. Add the labels **Yes** and **No** to the connectors leading from **Cancel?** To add labels, right-click each individual connector, and select **Add Label**. Rename labels to **Yes** and **No**:



2. Add the labels **Yes** and **No** to the connectors leading from **Approved?** Refer to the image above to see the labels.

3. Click **File > Save & Publish**.

Appian
Academy

Appian Step-by-Step #9

Exercise to Accompany
Process Modeling 101: Automate Your Business Processes

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 9: Process Modeling 101, Part 2

In this exercise, you will create a process model for requesting vehicle maintenance, along with other objects. You will create both the process model and related interfaces to ensure that they function correctly and meet application requirements. These requirements are:

- The process allows Mechanics to fill out the Request Maintenance form
- The process allows the Supervisor to approve maintenance requests
- If the maintenance request is approved, the process writes all vehicle details to the database

Before you start building the process model, let's create the interfaces and other objects that this process model will need.

Create the Maintenance Custom Data Type

Since this process model involves writing maintenance data to the database, you will need a maintenance data entity. You will use this maintenance data entity in both your interfaces and process model.

Follow the steps below to create a maintenance data entity.

1. In your application, click **New**, then **Data Type**.
2. Select the **Create from database table or view** radio button.
3. Select **Appian (Tomcat)** as the **Data Source** and **aamaintenance** as the **Table or View**.
4. Name the data type **AX_Maintenance**, and add a description: "Base data structure for a single Maintenance record."
5. Edit the field names to use camel casing.
6. Scroll down, and if necessary, rename the **Entity Name** to **AX_Maintenance**.
7. Click **Create**.

Create the Request Maintenance Form

In this part, you will create the AX_RequestMaintenance form that mechanics will use to request maintenance. This interface will contain a section with read-only vehicle details and an editable section for Mechanics to enter the maintenance-related details. At the end of this exercise, your interface will look like the image below.

Follow the steps below to create the request maintenance form.

Add Read-Only Vehicle Details

1. In your application, click **New** and create a new interface **AX_RequestMaintenance**. Add a simple description, and save this interface in the **AX Interfaces** folder.
2. Select the **One-Column Form** template, and update the form label to **New Maintenance Request**.
3. In the **Rule Inputs** pane in the top right corner, click the **plus** sign and add two rule inputs. Name the first rule input **vehicle**, and select the **AX_Vehicle** CDT as the **Type**:

4. Name the second rule input **maintenance**, and select the **AX_Maintenance** CDT as the **Type**.
5. Select the first section, and type **Vehicle Details** in the **Label** field.
6. Drag and drop three **Text** fields for the **Make**, **Model**, and **VIN** into the **Vehicle Details** section. Additionally, add two **Integer** fields for the **Year** and **Mileage**. Update the labels for each field.

7. For each field, configure the values in the **Display Value** and **Save Input To** fields. For example, use the dropdown menu for the Year field to select ri!vehicle.vehicleYear in both the Display Value and Save Input To fields.

8. For each field, change the label position to **Adjacent**, and select the **Read-only** checkbox.

9. Rearrange the fields to match the image below.

Add User Input Maintenance Details

1. Select the bottom section, and rename it to **Maintenance Details**.

2. From the **Components Palette**, drag and drop a **Columns** layout into the **Maintenance Details** section. Delete **one column**.

3. Drag and drop a **Date** component into the first column.

    - Update the label to **Request Date**.
    - Use the dropdown menu to select **ri!maintenance.maintenanceRequestDate** in the **Display Value** and **Save Input To** fields.
    - Select the **Required** checkbox.

4. Drag and drop a **Radio Buttons** component into the first column under Request Date.

    - Update the label to **Type of Maintenance**.
    - Click the **Expression Editor** icon next to **Choice Labels**, and type **{"Scheduled", "Unscheduled"}**.
    - Click the **Expression Editor** icon next to **Choice Labels**, and type **{true, false}**.
    - Select **ri!maintenance.maintenanceScheduled** from the dropdown for the **Display Value** and **Save Input To** fields.
    - Select the **Required** checkbox.
    - Scroll down to the **Choice Layout** field, and select **Compact**.

5. Drag and drop a **Paragraph** component into the second column.

    - Update the label to **Issue**.
    - Select **ri!maintenance.maintenanceIssue** from the dropdown for the **Display Value** and **Save Input To** fields.
    - Select the **Required** checkbox.

6. Let's add some testing values to this interface. Click **Test**, and then type **rule!AX_GetVehicleByID(1)** into the **Expression** field for the vehicle. Click **Set as Default Test Values**, then **Test Interface.** You will see that the interface is now pre-populated with vehicle values.

7. Click **Save Changes**.

Create the Approve Maintenance Form

In this part, you will create the AX_ApproveMaintenance interface for the AX Request Maintenance process model. Once a maintenance request is submitted, Supervisors will use this form to approve or deny the maintenance request. Your goal is to create an interface with one section for read-only vehicle details and one for read-only maintenance details. At the end of this part, your interface will look like the image below.

Follow the steps to create the approve maintenance form.

1. In your application, click **New**, and select **Interface**.

2. Select the **Duplicate existing interface** radio button, and then select the **AX_RequestMaintenance** interface.

3. Name this interface **AX_ApproveMaintenance**, and add a description: "Supervisor form for approving or denying a vehicle maintenance request." Save it in **AX Interfaces**, then click **Create**.

4. In the **Rule Inputs** pane, rename the **cancel** rule input to **approvalDecision**. Leave **Boolean** in the **Type** field.

5. Navigate to the **Maintenance Details** section, and select the **Request Date** field. Make the following changes to this field:

    - Change the **Label Position** to **Adjacent**.
    - Deselect the **Required** checkbox.
    - Select the **Read-only** checkbox.

6. Implement the same changes for the **Issue** field.

7. Delete the **Type of Maintenance** field, and replace it with a **Text** component. Make the following changes to the new text field:

    - Update the label to **Type of Maintenance**.
    - Set the **Label Position** to **Adjacent**.
    - Click the **Expression Editor** icon next to **Display Value**, and type the following expression:

        ```
        if(ri!maintenance.maintenanceScheduled, "Scheduled",
        "Unscheduled")
        ```

        This expression formats the field, so that true values display as "Scheduled" and false values display as "Unscheduled."

    - Select the **Read-only** checkbox.

8. Click the **Cancel** button, and make the following changes:

    - Update the label to **Deny**.
    - Under **Value**, replace **true** with **false**.
    - Make sure the **Submit** checkbox is selected.

9. Click the **Submit** button, and make the following changes:

    - Update the label to **Approve**.
    - Click the **Expression Editor** icon next to the **Value** field, and type **true**.
    - In **Save Value To**, select **ri!approvalDecision**.
    - Make sure the **Submit** checkbox is selected.

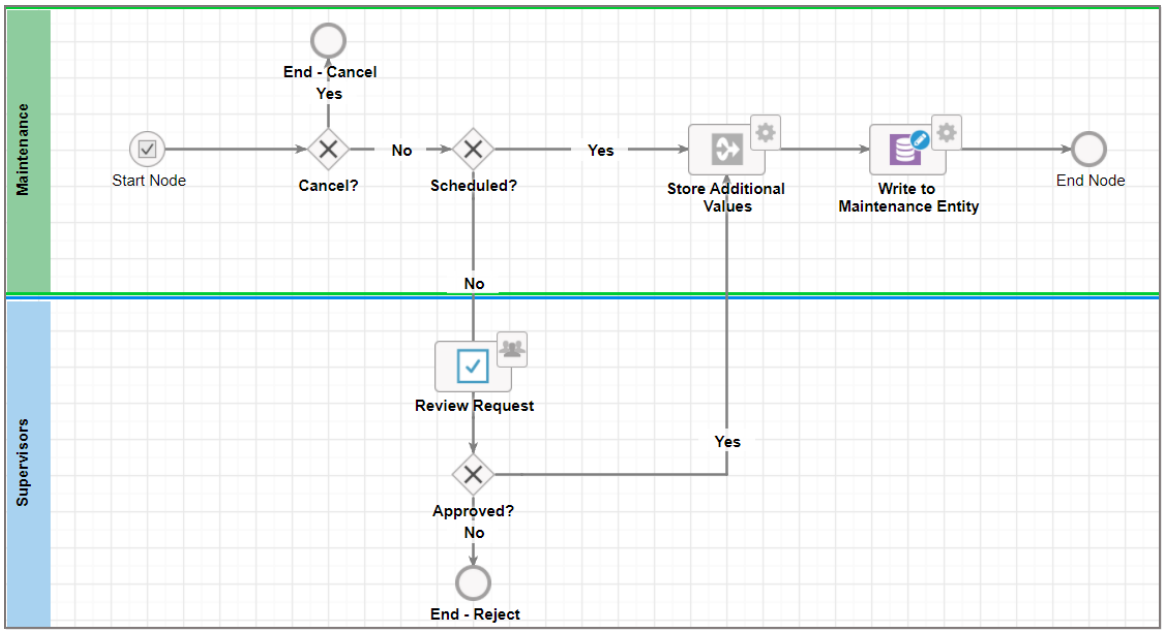
10. Click **Save Changes**.

Create the Request Maintenance Process Model

In this section, you will create a new process model that will automate the steps for requesting vehicle maintenance at Acme Automobile. The process starts with a Mechanic filling out a maintenance request. Once a request is submitted, the process checks whether a given request is a scheduled maintenance event or an unscheduled event that requires a Supervisor's

approval. While a scheduled maintenance event is written to the database without any further delay, an unscheduled request is first sent to a Supervisor for approval.

At the end of this exercise, your process model will look like the image below.



Create the Process Model

1. In your application, click **New** and select **Process Model**. Name this process **AX Request Maintenance**, and add a brief description: "Process model for requesting maintenance for a vehicle in the fleet." Click **Create**.

2. Unlike with the auto-generated process model, you will be prompted to secure this new process. Add **AX Administrators** as **Administrators** and **AX All Users** as **Viewers**. Click **Save**.

3. Add the swimlanes for Maintenance and Supervisors. Click the **Add Horizontal Lane** button on the toolbar  to add two swimlanes for Maintenance and Supervisors. Click **Lane 1** and **Lane 2** labels, and rename them to **Maintenance** and **Supervisors**.

Configure Process Properties

Now you can start configuring the process model properties, such as variables, alerts, and data management. Follow the steps below to configure the process properties.

1. Click the **Properties** button  in the toolbar.

2. Navigate to the **Process Start Form** tab. Type and select the **AX_RequestMaintenance** interface. Click **OK**, then **Yes** to automatically create process parameters.

3. Navigate to the **Variables** tab. You need to add a new process variable **approvalDecision** that you will use to configure the Approved? XOR node later. Click **Add Variable**, and name the new variable **approvalDecision**. Type **Boolean** into the **Type** field, and click **OK**.

4. Navigate to the **General** tab, and click the **Expression Editor** icon next to the **Process Display Name** field. You will now edit the display name to show the specifics of each maintenance request. Enter the following expression:

   ```
   "Request Maintenance: " & pv!vehicle.vehicleYear & " " &
   pv!vehicle.vehicleMake & " " & pv!vehicle.vehicleModel
   ```

   Insert the three process variables by selecting them from the **Data** tab. Click **Save and Close.**

5. Navigate to the **Alerts** tab. Select the **Use custom error alert settings** radio button. Also, select the **Send alerts to the following users & groups** checkbox, and type **AX Administrators** into the adjoining field**.**

6. Navigate to the **Data Management** tab. Select **Archive processes**, and change it to **3** days.

7. Click **OK**, then **File > Save & Publish**.

8. Now that you have wired the form to the process, test your process model to check that everything works correctly and the process advances to completion. Follow the steps below to test the process:

   - Click **File > Start Process for Debugging**.
   - Complete the form, and click **Submit**. You will notice that a new process instance started in the **Monitoring Mode** and on a new tab in the Process Modeler.

- Click the **Refresh** button  in the toolbar to update the process flow. You will notice that the process will advance to the **End Node**:

- Click the **Process Details** button  in the toolbar, and navigate to the **Variables** tab to verify that the data you entered into the form was captured by the process model.

- Close the **Monitoring** tab, but keep the Process Model open to configure our next sequence of activities.

As you continue working on your process model, test frequently to ensure that the nodes are configured correctly, and all data is captured as expected.

Configure the Cancel XOR Gateway

In this process, if a Mechanic cancels the start form, the process will end. If a Mechanic submits the form, the process will continue to the next process node. Follow the steps below to configure the Cancel? gateway.

1. Drag and drop an **XOR Gateway** into the Maintenance swimlane to the right of the Start Node. Rename it to **Cancel?**

2. Drag and drop an **End Event** into the Maintenance swimlane above Cancel?. Rename it to **End - Cancel**. Connect the **Cancel?** and **End - Cancel** nodes.

   To connect, hold down **SHIFT** on your keyboard, and then click from one node to another. SHIFT will automatically toggle your pointer to the connector tool. You can also use the **Connect** icon  to connect the nodes.

3. Drag and drop an **XOR Gateway** into the Maintenance swimlane to the right of Cancel? Rename it to **Scheduled?**

4. Double-click the **Cancel?** gateway, and navigate to the **Decision** tab.

5. Click the **New Condition** button, and then click the **Expression Editor** icon next to the first condition. Select the **cancel** process variable from the **Data** tab, or simply type **pv!cancel**. Click **Save and Close**.

6. For the first condition, select **End - Cancel** in the **Result** column. Put **Yes** in the **Path Label** column.

7. For the second condition, select **Scheduled?** in the **Result** column.

8. Click **OK.** Add the label **Yes** to the connector leading from **Cancel?** to **End - Cancel**, and the label **No** to the connector leading from **Cancel?** to **Scheduled?** To add labels, right-click each individual connector, and select **Add Label**.

9. Click **File > Save & Publish**.

Configure the Scheduled XOR Gateway

Next, configure the Scheduled? XOR gateway. This gateway will direct scheduled maintenance events to the Store Additional Values node and unscheduled events to the Review Request node. Follow the steps below to configure this gateway.

1. Drag and drop a **User Input Task** into the Supervisor swimlane below Scheduled? Rename it to **Review Request**. Connect the **Scheduled?** and **Review Request** nodes.

2. Drag and drop an **XOR Gateway** into the Supervisor swimlane below Review Maintenance Request. Rename it to **Approved?** Connect the **Review Request** and **Approved?** Nodes.

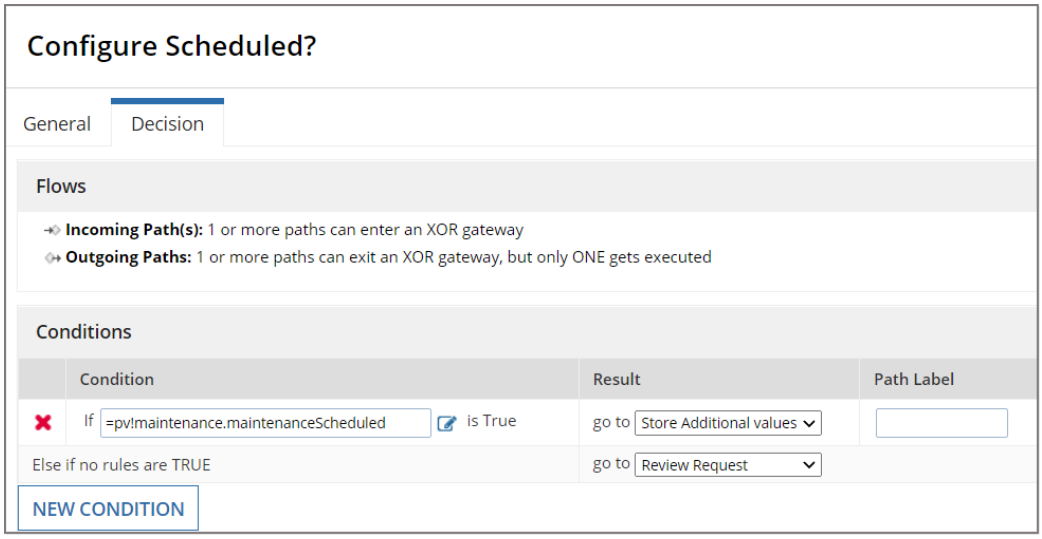
3. Drag and drop a **Script Task** into the Maintenance swimlane to the right of Scheduled? Rename it to **Store Additional Values**. Connect the **Approved?** and **Store Additional Values** nodes.

4. Double-click **Scheduled?**, and navigate to the **Decision** tab.

5. Click the **New Condition** button, and then click the **Expression Editor** icon next to the first condition. Click the **+** sign next to **maintenance** to expand it, and then select the **maintenanceScheduled** process variable. The expression should now be populated with **pv!maintenance.maintenanceScheduled**. Click **Save and Close**.

6. For the first condition, select **Store Additional Values** in the **Result** column. Put **Yes** in the **Path Label** column.

7. For the second condition, select **Review Request** in the **Result** column.



8. Click **OK**. Add the label **Yes** to the connector leading from **Scheduled?** to **Store Additional Values**, and the label **No** to the connector leading from **Scheduled?** to **Review Request**. To add labels, right-click each individual connector, and select **Add Label**.

9. Click **File > Save & Publish**.

Configure the User Input Task

Next, let's configure the Review Request node. You will use this node to add the Approve Maintenance form to this process. Complete the steps below.

1. Double-click **Review Request**. First, edit the **Task Display Name** to make it dynamic. Click the **Expression Editor** icon, and select the process variables for the **year**, **make**, and **model** to sert them into the expression. Your expression should look as follows:

```
"Review Maintenance: " & pv!vehicle.vehicleYear & " " &
pv!vehicle.vehicleMake & " " & pv!vehicle.vehicleModel
```

   Click **Save and Close**.

2. Navigate to the **Forms** tab. Type and select **AX_ApproveMaintenance** as the interface. Click **Yes** to automatically create node inputs.

3. Navigate to the **Assignment** tab. Assign this node to **AX Supervisors**.

4. Navigate to the **Data** tab. Under **Inputs**, click the first node input **vehicle**. For the **Value** field, select **vehicle**.

5. Click the second node input **maintenance**. For the **Value** field, select **maintenance**.

6. Click the third node input **approvalDecision**. For the **Save into** field, select **approvalDecision**. Note that you make your selection under the **Save into** field because this is the new information passed back from the user that you want to write to the database table.

7. Click **OK**, then **File > Save & Publish**.

Configure the Approved Gateway

If the Supervisor approves the maintenance request, the process will continue to the script task. If the Supervisor denies the maintenance request, the process will end. Follow the steps below to configure the Approved? gateway added earlier.

1. Drag and drop an **End Event** into the Supervisor swimlane below **Approved?** Rename it to **End - Reject**, and then connect these two nodes.

2. Double-click **Approved?**, and navigate to the **Decision** tab.

3. Click the **New Condition** button, and then click the **Expression Editor** icon next to the first condition. In the **Data** tab, select the **approvalDecision** process variable. Click **Save and Close**.

4. For the first condition, select **Store Additional Values** in the **Result** column. Put **Yes** in the **Path Label** column.

5. For the second condition, select **End - Reject** in the **Result** column.

6. Click **OK**. Add the label **Yes** to the connector leading from **Approved?** to **Store Additional Values**, and the label **No** to the connector leading from **Approved?** to **End - Reject**. To add labels, right-click each individual connector, and select **Add Label**, or double-click the connector to edit the label in the **Flow Properties**.

7. Click **File > Save & Publish**.

Configure the Script Task

For this process, you will use a script task to store six variables: Maintenance Last Modified Date, Maintenance Review Date, Maintenance Requester, Maintenance Last Modified By, Maintenance Status, and Vehicle ID. Follow the steps below to set up the Store Additional Values script task.

1. Double-click **Store Additional Values**, and navigate to the **Data** tab. Click the **Outputs** tab.

2. Click **New Custom Output**. You will want to set the **Maintenance Last Modified Date** as the current date. Click the **Expression Editor** icon, and type the function **today()**. Click **Save and Close**. For the **Target**, click the **inline arrow**, and hold your cursor over **maintenance**. Select **maintenance.maintenanceLastModifiedDate**.

3. Repeat the same steps for the **Maintenance Review Date**. Be sure to select **maintenance.maintenanceReviewDate** as the **Target**.

4. Click **New Custom Output**. You will now want to set the **Maintenance Requester** as the initiator of the process model. Click the **Expression Editor** icon, and type the function **pp!initiator**. Alternatively, click **initiator** under **Process Properties**. Click **Save and Close**. For the **Target**, select **maintenance.maintenanceRequester**.

5. Repeat the same steps for **Maintenance Last Modified By**. Be sure to select **maintenance.maintenanceLastModifiedBy** as the **Target**.

6. Click **New Custom Output**. You will now want to set the **Maintenance Status** as Scheduled. Click the **Expression Editor** icon, and type **"Scheduled"**. Click **Save and Close.** For the **Target**, select **maintenance.maintenanceStatus**.

7. Click **New Custom Output**. The last value to save is the **Vehicle ID** associated with the maintenance request. Click the **Expression Editor** icon, and then find **vehicle** under **Process Variables** on the left. Click the plus sign, then select **vehicleId**. The expression should be populated to look like this:

   ```
   pv!vehicle.vehicleId
   ```

   Click **Save and Close**. For the **Target**, select **maintenance.vehicleId**.

8. Click **OK**, then **File > Save & Publish**.

9. Test your process model using the steps below:

   - Click **File > Start Process for Debugging**.
   - Complete the **New Maintenance Request** form, selecting the unscheduled maintenance, and click **Submit**.
   - In the new **Monitoring** tab, click the **Refresh**  button in the toolbar.
   - Right-click the **Review Request** task once it is highlighted in green, and select **View Form.**
   - Accept the task, and then click **Approve**.
   - Click **Refresh** in the toolbar. You will see the process advance to the **End Node**.
   - Click **Process Details**, and select the **Process Variables** tab to view the data that was collected.

- Repeat the testing steps, but using different selections. For example, request an unscheduled maintenance, and then click **Deny** instead of Approve. Make sure that the process flows correctly.

Configure the Write to Maintenance Entity Service

Now that you have two forms and a script task in your process, add and configure the Write to Data Store Entity smart service to ensure that all maintenance data is written to the database table. Follow the steps below to complete this task.

1. Drag and drop a **Write to Data Store Entity** node into the **Maintenance** swimlane, to the right of Store Additional Values. Rename it to **Write to Maintenance Entity**.
2. Double-click **Write to Maintenance Entity**.
3. Navigate to the **Data** tab, and then the **Inputs** tab.
4. Click **Data Store Entity**, and then the **Directory icon**  next to the **Value** field.
5. Click **AX Data Store**, then select **AX_Maintenance**. Click **OK**.
6. Click **New Input**, and name it **maintenance**.
7. In the **Type** field, type and select **AX_Maintenance** once it appears.
8. In the **Value** field, use the dropdown arrow to select the process variable **maintenance**.
9. Navigate to the **Assignment** tab. Select the **Run as whoever designed this process model** radio button, and click **OK**.
10. Click **File > Save & Publish**.

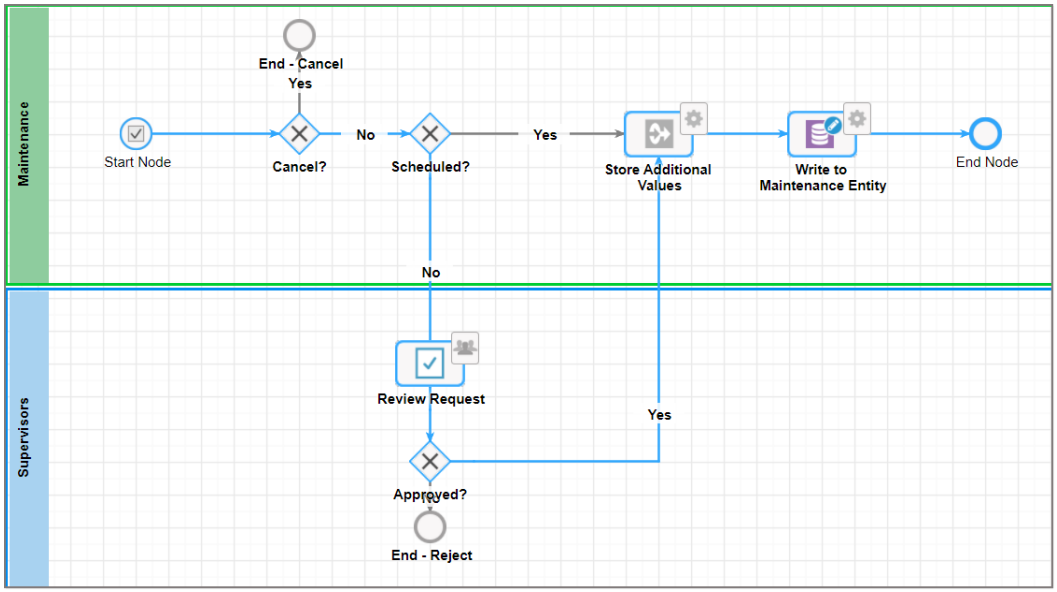
Test the Process

Let's test this process model one more time to ensure that it's working as expected. Keep in mind that when you work on an actual project, testing of this kind should be performed each time you add a new node to the process model. Follow the steps below to test the process model.

1. Click **File**, and select **Start Process for Debugging**.
2. Complete the **Request Maintenance** form, and click **Submit**.
3. Click the **Refresh** button  in the toolbar to update the process flow. If you choose **Unscheduled** as the Maintenance Type, you will notice that the process will advance to the **Review Maintenance Request** node.

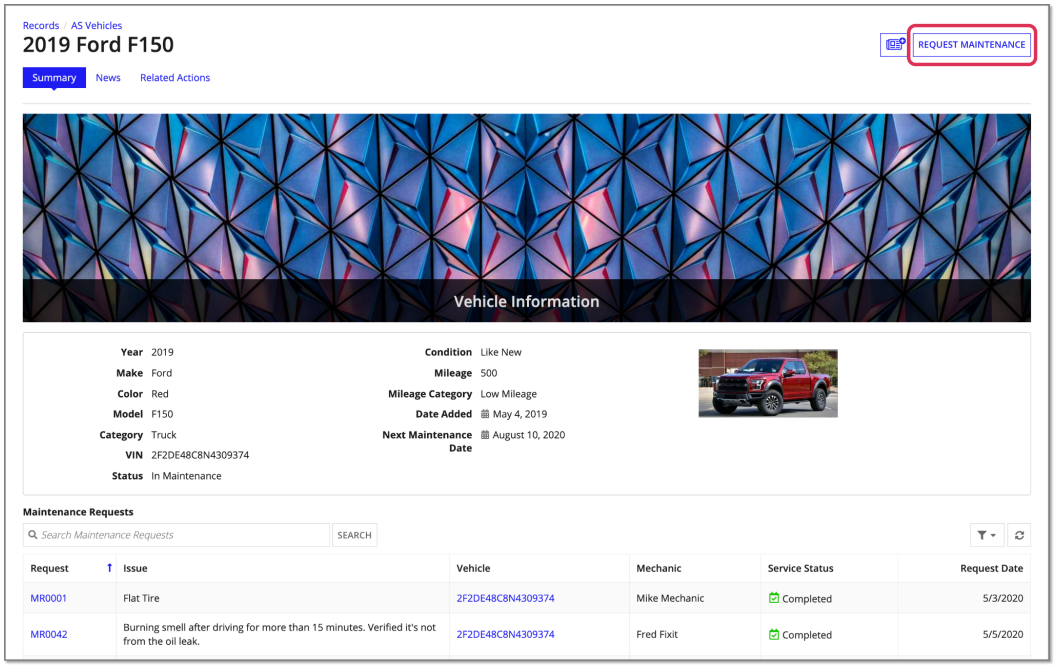
4. Right-click the **Review Maintenance Request** node, and select **View Form**. Click **Accept** to accept the task, and then click **Approve**.

5. Click the **Refresh** button  once again to update the process flow. You will notice that the process will advance to the **End - Approve** node:



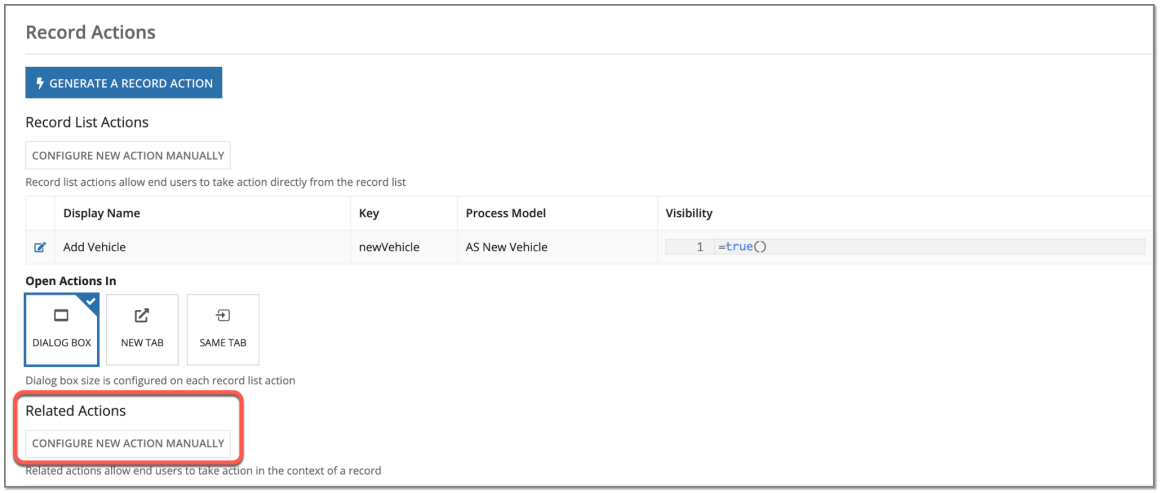
Create a Related Action

In this exercise, you will create a related action that will enable mechanics to request maintenance from the summary view for a specific vehicle.



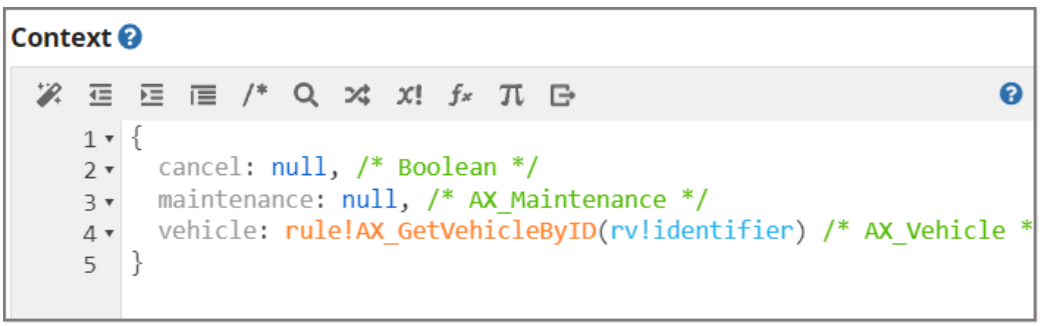
You will create this related action by adding the AX Request Maintenance process model to the AX Vehicle record. Follow the steps below to create a related action.

1. In the **AX Vehicle** record, navigate to the **Record Actions** tab.
2. Under **Related Actions**, click **Configure New Action Manually**:



3. Type **Request Maintenance** into the **Display Name** field, and add a simple description, "Request maintenance for a vehicle."
4. Select **Medium** for **Dialog Box Size**.
5. Select **AX Request Maintenance** for **Process Model**.
6. The Context variables will be generated for you. You use the Context box to pass in record information into the process model backing the related action. Note that the generated variables are the process variables from **AX Request Maintenance** that are set as parameters. Leave **cancel** and **maintenance** as **null**.
7. In the **Context** box, replace the **null** next to **vehicle** with the following expression:

```
rule!AX_GetVehicleByID(rv!identifier)
```



rv!identifier references a record's unique ID, which should always be the same as the primary key ID for the database table backing the record. In this case, since we are in

the AX Vehicle record, this expression uses the unique vehicleId to pass vehicle information into the process model.

8. Now, you have to set the visibility of the related action. In the **Visibility** expression editor, type the following expression:

```
a!isusermemberofgroup(loggedInUser(),
cons!AX_MAINTENANCE_POINTER)
```

   This expression checks whether the logged in user is included in the AX Maintenance group or not. This related action is now only visible if the user is in the Maintenance group. Click **OK**.

9. Let's add the Related Action as a shortcut:

   - Navigate to the **Views** tab on the left hand side.
   - Click the **Edit** icon next to **Summary**.
   - Select the **Request Maintenance** checkbox under **Related Actions Shortcuts**.
   - Click **OK**, then **Save Changes**.

10. To test this feature from the business user perspective, click the link located next to the record name. Select any vehicle. The Request Maintenance related action button will be in the upper right corner. Test the flow to see how it works.

Appian
Academy

Appian Step-by-Step #10

Exercise to Accompany
Reports: Build Basic Charts and Grids

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 10: Reports

In this exercise, you will create a report for Supervisors and Registrars. This report will contain:

- A column chart of vehicles by make and category
- A pie chart of vehicles by make
- A drillable bar chart by make

You will create each chart as a separate interface. After, you will combine them into a single master interface. This master interface will be used to create a report object that you will later add to your application site.

Create a Column Chart

Let's create the column chart of vehicles by make and category first.

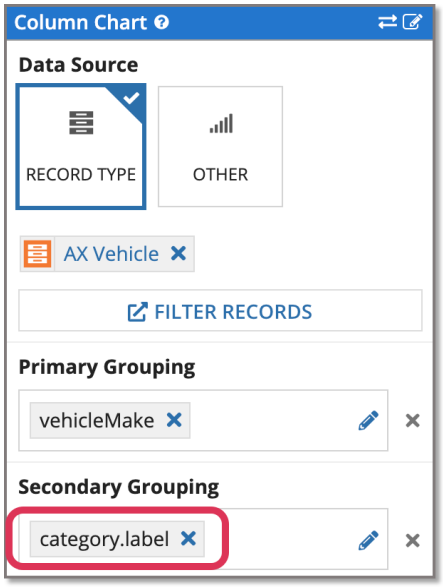
1. In your application, click **New**, and then **Interface**. Name the interface **AX_VehicleColumnChart**, and add a description, "An interface with the column chart of vehicles by make and category." Save it in **AX Interfaces**, and click **Create**.

2. In the **Components** palette, scroll down to **Charts**. Drag and drop a **Column Chart** component onto the canvas. Click the title **Column Chart**, and rename it to **Vehicles by Make and Category**.

3. In the **Component Configuration** pane, select **Record Type** as the **Data Source**. Start typing **AX Vehicle** into **Search record types**, and then select it from the list of auto-suggestions.

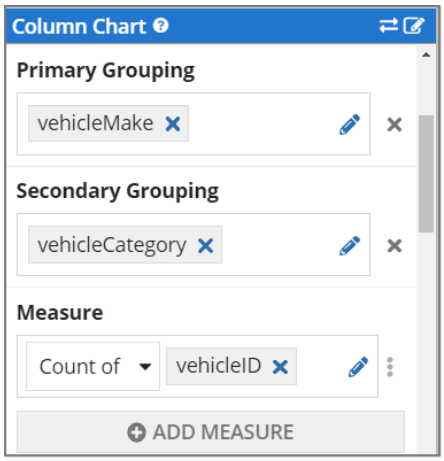
4. Once you select AX Vehicle , you will notice that the primary grouping as well as measure properties get auto-populated. The primary grouping should be by **vehicleMake**. If your chart has a different field, remove that value, and select **vehicleMake**.

5. Next, you will configure the **Secondary Grouping** property. Click **Add Grouping**, and click the arrow for the dropdown menu in the **Secondary Grouping** field. Scroll all the way down to the bottom of the dropdown menu. Hover over **category**, and then select **label**.

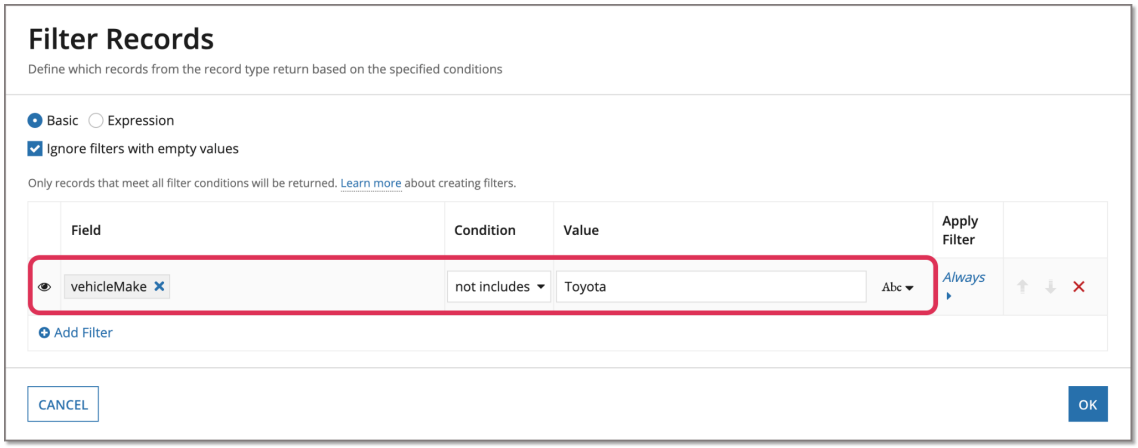


6. Scroll down to **Stacking**, and select **Normal** from the dropdown menu. Then, select the **Show data labels** check box.

7. Now, scroll up to **Measure**. This field got auto-populated correctly with the count of vehicle IDs. Keep the auto-populated value.



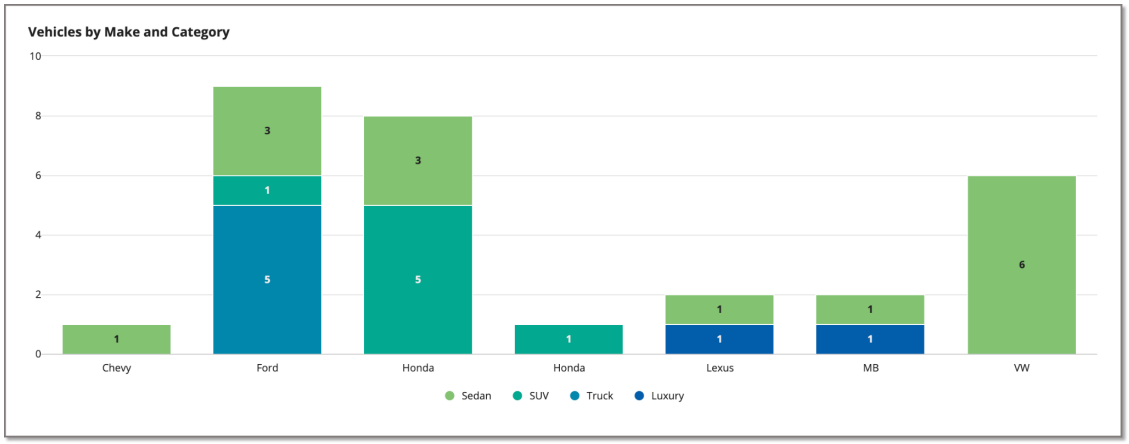
8. Next, add a record filter. Record filters are helpful when you want to show some, but not all record data. For example, if you want to hide Toyotas in the chart, click **Filter Records** at the top of the **Component Configuration** pane, and select **vehicleMake** as the **Field**. Select **not includes** in the **Condition** column. Type **Toyota** into the **Value** field.



Click **OK**. You will notice that Toyotas are now hidden.

9. Scroll down to **Color Scheme**, and select **Rainforest**. Note that you can also adjust the height of this chart. This selection is available below the **Color Scheme** menu.

10. Click **Save Changes**. Refer to the image below to see the final column chart:



Create a Pie Chart

Next, let's create a pie chart of vehicles by category.

1. In your application, create another interface. Name it **AX_VehiclePieChart**, and add a brief description: "An interface with the pie chart of vehicles by make." Save it in **AX Interfaces**. Click **Create**.

2. In the **Components** palette, scroll down to **Charts**. Drag and drop a **Pie Chart** component onto the canvas. Click the title **Pie Chart**, and rename it to **Vehicles by Make**.

3. In the **Component Configuration** pane, select **Record Type** as the **Data Source**. Start typing **AX Vehicle** into **Search record types**, and then select this record from the list of auto-suggestions.

4. Once you select AX Vehicle, you will notice that the primary grouping and measure properties auto-populate. The primary grouping should be by **vehicleMake**. If your chart has a different field, remove that value, and select **vehicleMake**.

5. Scroll down to **Series**, and select the checkboxes for **Show data labels** and **Show as percentage**.

6. Next, sort your chart from largest to smallest percentage. Scroll to the **Sort** property, and click **Add Sort**. Select **vehicleId_count_measure1** from the **Sort By** dropdown menu. Select the **Descending** radio button as the **Order**.

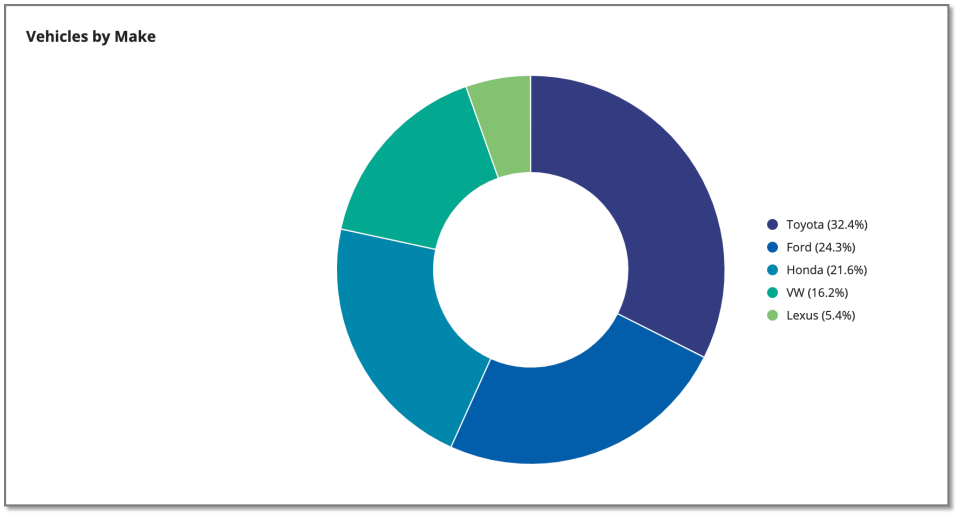
7. Scroll to the **Data Limit** property field. This property determines how many unique groupings are shown on the chart. As a best practice, pie charts are best used when the number of unique groupings is less than 5. Type **5** into **Data Limit** to limit this pie chart to the top five makes in the Acme Automobile fleet.

8. Scroll down to **Color Scheme**, and select **Rainforest**.

9. Scroll to **Style**, and note that you can toggle between the Donut and Pie styles for this chart.

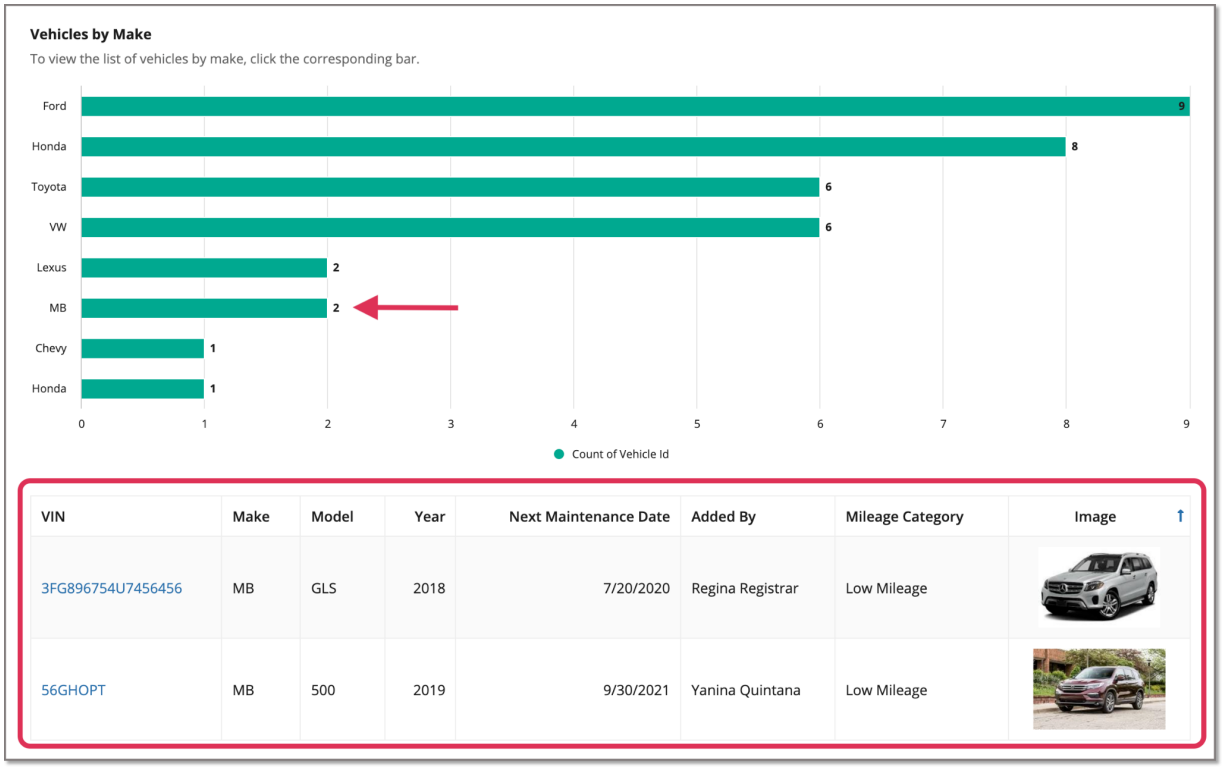
10. Scroll to **Series Labels**, and select **In Legend**.

11. Click **Save Changes**. Refer to the image below to see the final pie chart:



Create a Drillable Bar Chart

In this step, you'll create a drillable bar chart of vehicles by make. If you click any of the bars in this drillable chart, the grid with the vehicles of the selected make will display right below the bar chart:



Follow the steps below to create the drillable chart.

1. In your application, click **New**, and select **Interface**. Name the interface **AX_VehicleDrillableChart**, and add a description, "An interface with the drillable bar chart of vehicles by make." Save this interface in **AX Interfaces**. Click **Create**.

2. In the **Components** palette, scroll down to **Charts**. Drag and drop a **Bar Chart** component onto the canvas. Click the title **Bar Chart**, and rename it to **Vehicles by Make**.

3. In the **Component Configuration** pane, select **Record Type** as the **Data Source**. Start typing **AX Vehicle** into Search record types, and then select it from the list of auto-suggestions.

4. Once you select AX Vehicle, the primary grouping as well as measure properties get auto-populated. Your primary grouping should be **vehicleMake**. If your chart has a different grouping, remove that value, and select **vehicleMake**.

5. Scroll down to **Instructions**, and add an instruction, "To view the list of vehicles by make, click the corresponding bar." This will ensure that users know that the bar chart is clickable.

6. Select the **Show legend** and **Show data labels** checkboxes. Update the **Color Scheme** to **Rainforest**.

7. Scroll up to the **Sort** property, and click **Add Sort**. Select **vehicleId_count_measure1** from the **Sort By** dropdown menu. Select the **Descending** radio button as the **Order**.

8. Click **Save Changes**.

Add a Read-Only Grid

1. Now it's time to build the grid. In the **Components** palette, drag and drop a **Read-Only Grid** underneath the bar chart.

2. In the **Component Configuration**, select **Record Type,** and then choose **AX Vehicle** as your target record.

3. Remove the default label **Read-only Grid**. Also, scroll down to **Records Actions**, and click **X** next to **Add Vehicle** to remove it.

4. Deselect the **Show refresh button** and **Show search box** checkboxes.

5. Next, let's add a record filter to this grid to ensure that vehicles display by make. This can be done after you add the **make** rule input. Click the **plus sign** next to **Rule Inputs**, and type **make** into the **Name** field, and **Text** into the **Type** field. Click **Create**.

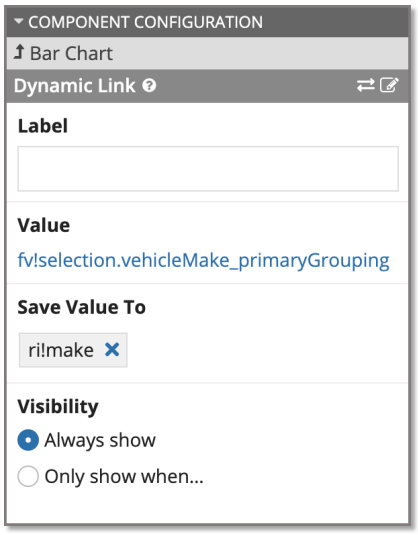
6. Click **Filter Records**. In the new dialog box, click **Add Filter**. Select **vehicleMake** in the **Field** column, and select **equal to** in the **Condition** column. In the **Value** column, click the **arrow** next to Abc, and select **Input/Variable** from the dropdown menu. In the **Value** dropdown, select **ri!make - Text**, and click **OK**.

7. Finally, let's configure the dynamic link that will link the bar chart to the grid. Select the bar chart in the canvas, and scroll to **Link** in the **Component Configuration** pane. Click **Insert Link**, and then click **Dynamic Link**.

8. Click **fv!selection**. Edit this expression to:

```
fv!selection.vehicleMake_primaryGrouping.
```

In the **Save Value To** field, select **ri!make**:



vehicleMake_primaryGrouping is the alias for the bar chart's Primary Grouping property. To look it up, go to **Primary Grouping** and click the **Expression Editor** icon next to it.

9. Click **Save Changes**. Your drillable chart is now fully configured. To test the chart, click any bar in the bar chart and see the list of vehicles in the grid dynamically update to the selected make.

10. To hide the grid when no make is selected, select the grid and scroll to the **Layout** menu. Expand it, and in **Visibility** select the **Only show when** radio button. Click **Edit Condition**, and enter the following expression: **not(isnull(ri!make))**. This expression will show the grid only if a make is selected.

11. Click **OK**, then **Save Changes**.

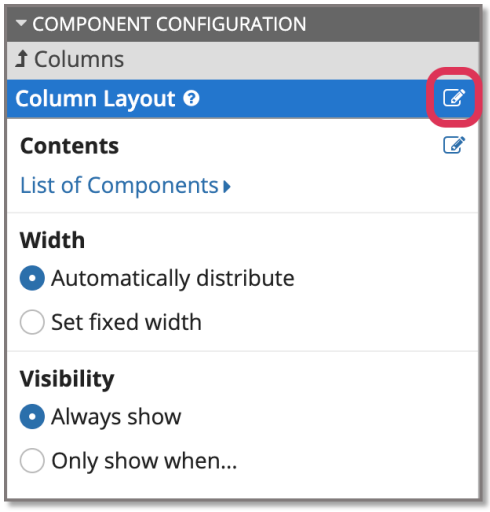
Create a Report Interface

Now, let's combine all three charts into a single interface.

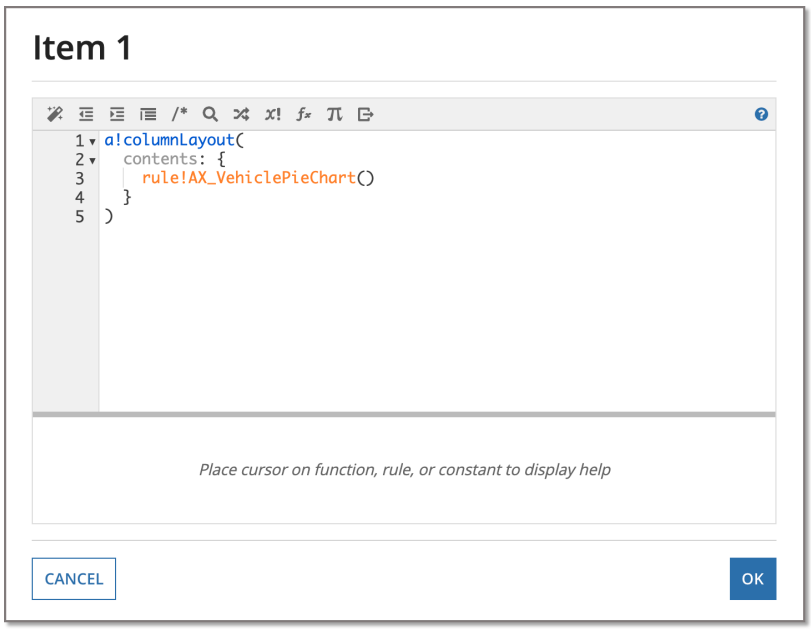
1. In your application, create a new interface. Name it **AX_VehicleReportInterface**. Add a brief description, "An interface that shows the charts of vehicles by category and make." Save it in **AX Interfaces**, then click **Create**.

2. In the **Components** palette, drag and drop a **Section** layout onto the canvas. Next, grab the **Columns** layout, and drop it inside the **Section** layout. Delete one column, and update the section label to **Fleet Data**.

3. Click the **first column** to select it, and then in the **Component Configuration** pane click the **Edit as Expression** icon located next to **Column Layout**:



4. In the **Expression Editor,** click within the curly brackets, and type **rule!AX_VehiclePieChart**. Click **OK**.



5. Click the **second column**, and then repeat the steps above, but this time type the following expression: **rule!AX_VehicleColumnChart**. Click **OK**.

6. Grab another **Section** layout from the **Component** palette, and drop it underneath the first section. Remove the default label for this section.

7. Next, let's add a local variable to this interface. In this exercise, we need a local variable to help us with holding the temporary make selection that we configured in the drillable bar chart.

   Local variables are added to an interface only in the Expression Mode, and they go at the very beginning of the interface expression, encapsulating the whole expression within parentheses. We will use the **a!localVariables** function to declare a local variable **local!selection**.

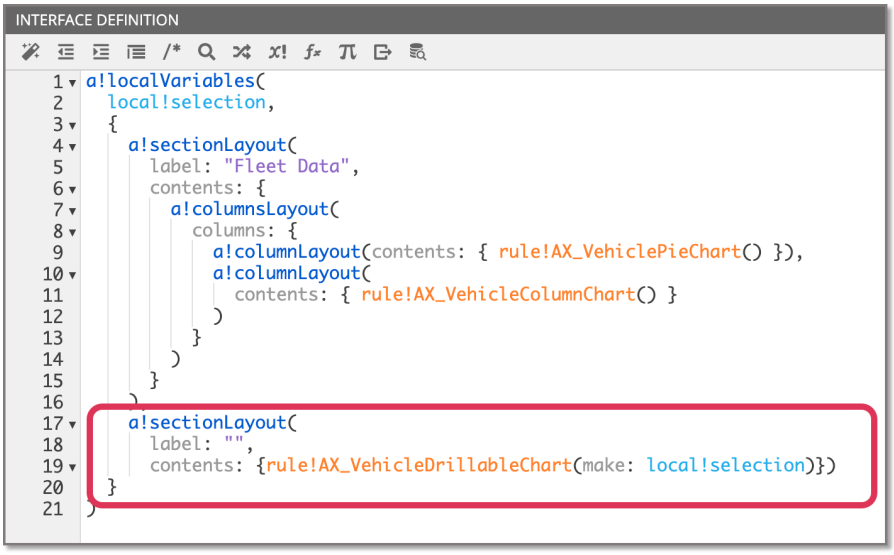
8. Click to access the **Expression Mode**:

9. Click before the curly bracket for the expression, and press **Enter**.

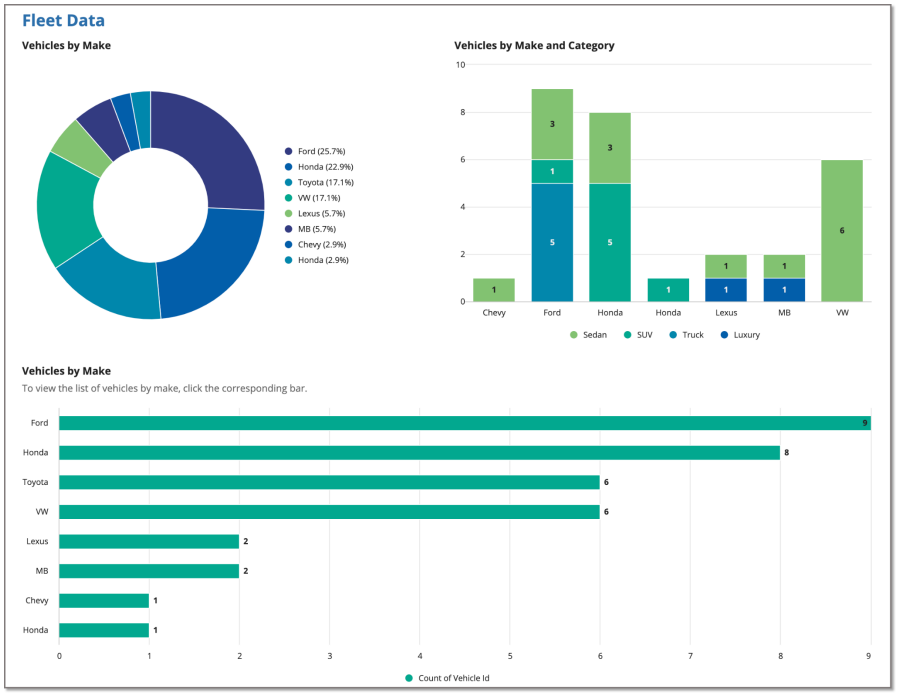
   - In line 1, start typing **a!localVariables**, and then select this option when it gets auto-suggested.
   - Since all local variables must be defined first in the code, create a new line right after a!localVariables, and type **local!selection**.
   - Add a **comma** after it, and delete the **closing parenthesis**.
   - Scroll to the end of the entire interface expression, and add the **parenthesis**.

10. Because we are already in the Expression Mode, let's add the drillable bar chart to the interface using an expression. Locate the last section that we added to this interface in the Design Mode, and click inside the curly brackets for **contents**. Type the following expression:

```
rule!AX_VehicleDrillableChart(make: local!selection)
```



11. Return to **Design Mode**, and click **Save Changes**. Your interface is now complete and ready to be used for the report object:



Create a Report Object

Now that you've combined the charts into a single interface, create a Report object. Follow the steps below to finalize your report.

1. In your application, click **New**, then **Report**. Name it **AX Fleet Report**, and add a brief description: "Report for Supervisors and Registrars that contains all relevant vehicle analytics." Select **AX_VehicleReportInterface** in the **Interface** field, and click **Create**.
2. Configure security for the report. Add the **AX All Users** as **Viewers** and **AX Administrators** as **Administrators**. Click **Save**.

Set this object aside for now. In the next exercise you'll add it as a page to your site.

Appian
Academy

Appian Step-by-Step #11

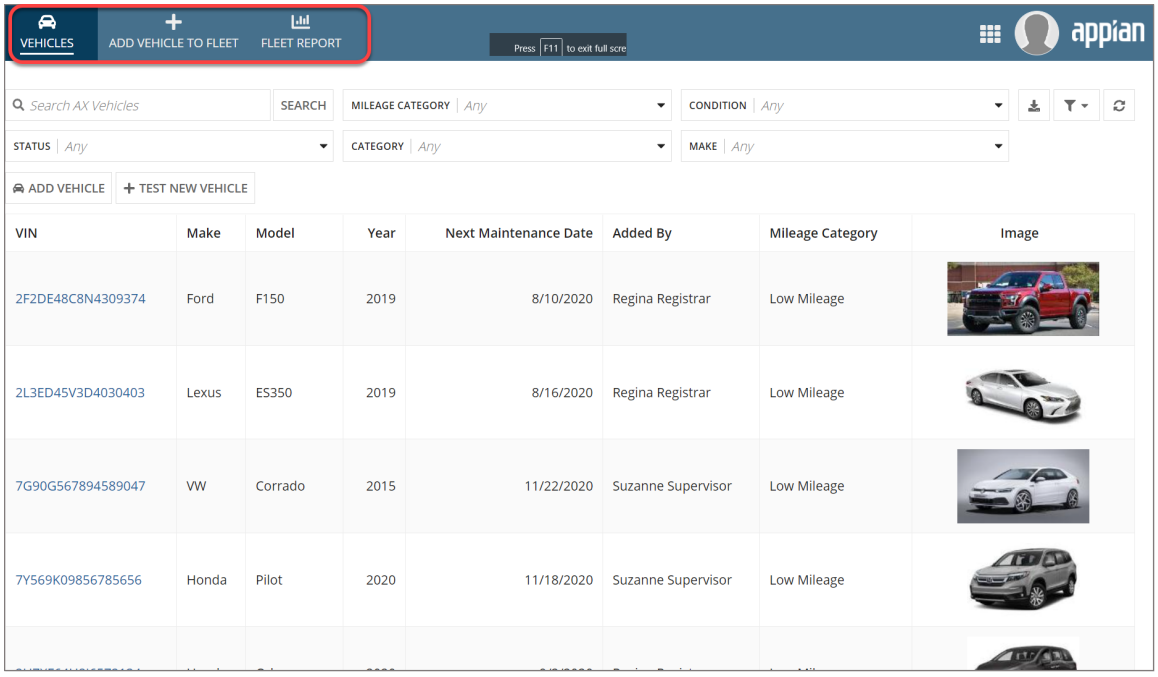
Exercise to Accompany
Sites: Create a Custom and Focused User Experience

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 11: Sites

In this exercise, you will create a site for the business users — Supervisors, Registrars, and Mechanics — where they will access their fleet-related information. This site will include the following pages:

- The page for the Vehicle record
- The page for adding vehicles to fleet
- The page with the Fleet Report



Create the Site

1. In your application, click **New**, then **Site**. Name it using the name of the app: **Acme Exercise Site**. Add a description, "Site for Acme Automobile employees," and click **Create**.

2. Configure security for the Site. Add the **AX Administrators** group as **Administrators** and **AX All Users** as **Viewers**.

Add the Pages

1. Add the **AX Vehicle** record. Click the **Add Page** button, and then configure the fields as follows:

    - Title the page **Vehicles**.
    - Click the dropdown menu for the icons, and then browse or search for an appropriate icon.
    - Select **Record List** in the **Type** field.

- Select the **AX Vehicle** record in the **Content** field. Click **OK**.

2. Add the **Add Vehicle to Fleet** action page. Click the **Add Page** button, and then configure the fields as follows:

    - Title the page **Add Vehicle to Fleet**.
    - Click the dropdown menu for the icons, and then browse or search for an appropriate icon.
    - Retain **Action** in the **Type** field.
    - Select the **AX New Vehicle** process model in the **Content** field. Click **OK**.

3. Finally, add the **Report** page. Click the **Add Page** button, and then configure the fields as follows:

    - Title the page **Fleet Report**.
    - Click the dropdown menu for the icons, and then browse or search for an appropriate icon.
    - Select **Report** in the **Type** field.
    - Select the **AX Fleet Report** in the **Content** field. Click **OK**.

4. Click **Save Changes**.

5. Before you close this object, check out your new site. Scroll all the way up, and click the **URL** in the **Web Address** field. Alternatively, click the Navigation menu  and select **Acme Exercise Site**. Test your new site by completing any or all of the following tasks:

    - Review the list of vehicles
    - Access a summary view for a vehicle or two, and check that the Request Maintenance button is available
    - Access the Add Vehicle to Fleet page to see the Add Vehicle form
    - Access the Fleet Report page, and review the charts. Test out the drillable chart!

Appian Academy

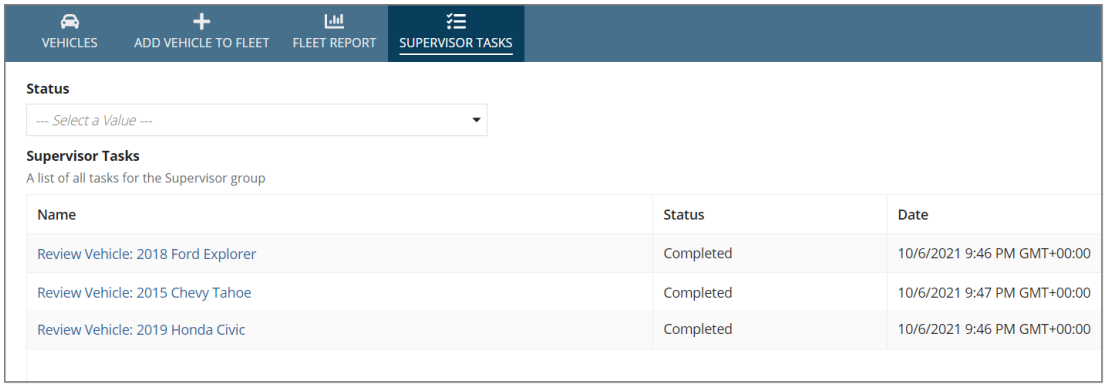
Appian Step-by-Step #12

Exercise to Accompany
Sites: Create a Custom and Focused User Experience

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

Exercise 12: Task Report

In this exercise, you'll learn how to create a Supervisor Task Report that will allow Supervisors to view and perform their vehicle approval tasks. Once your task report is assembled, you'll add it as a page to the AX site. Supervisors will use this page to view tasks, filter them by status, and navigate to the UIs holding specific tasks by clicking the hyperlinked task name. Your final site page will look as follows:



To build a task report, you'll start by building a process report that will help you to pull data from the AX New Vehicle process model. At a high-level, you'll complete these steps to create the task report for Supervisors:

- Create a process report
- Create a constant to point to the process report
- Build an interface to display tasks, link tasks to individual approval forms, and add a Status filter to this interface
- Create a task report
- Add the task process report as a page to the Acme Exercise Site

Follow the steps below to build a task report.

Create a Process Report

Let's create the task process report to show the tasks that are assigned to the Supervisor group:

1. From within the **AX Document** folder, click **New**, and select **Process Report**.

2. Retain the **Duplicate existing process report** selection, and search and select **My Tasks** in the **Process Report to Duplicate** box. If no suggestions appear for **My Tasks**, use **active_tasks** instead.

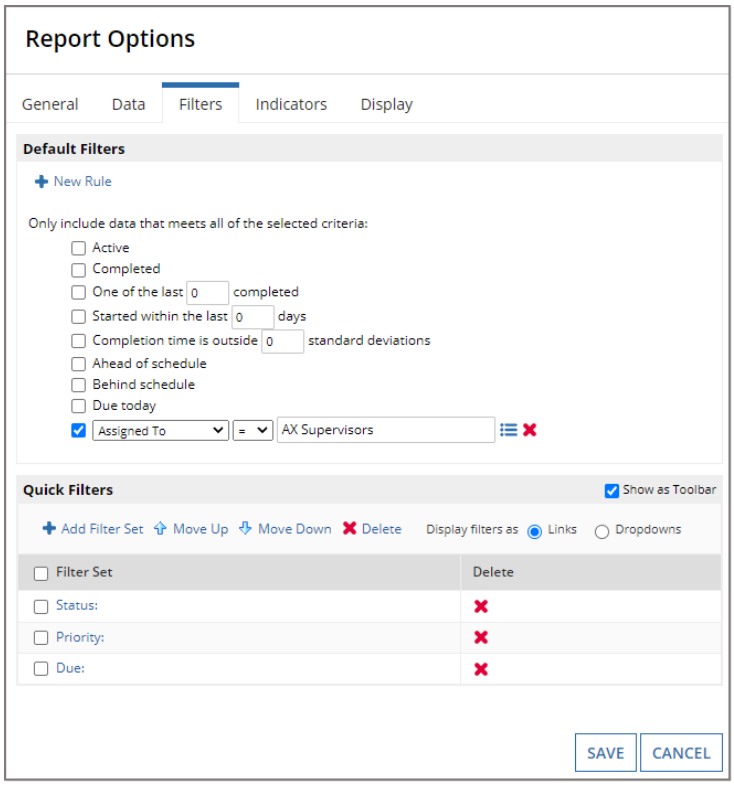
3. Name it **AX Supervisor Process Report**, and then change the description to "A list of tasks for the AX Supervisors group."

4. Make sure the **Save In** folder is set to **AX Documents**, and click **Create**.

Modify the Process Report

In the previous step, you created the AX Supervisor Process Report by duplicating the out-of-the-box My Task report, which by default displays all tasks for the logged in user. In this section, you'll modify this process report to display tasks assigned to the AX Supervisor group:

1. Locate and open the **AX Supervisor Process Report**.
2. On the top menu bar, select **All** next to **Status**, and then click **Edit**.
3. Navigate to the **Filters** tab, and click **New Rule**. In the first dropdown menu, select **Assigned To**, and then type **AX Supervisors** in the text entry field.

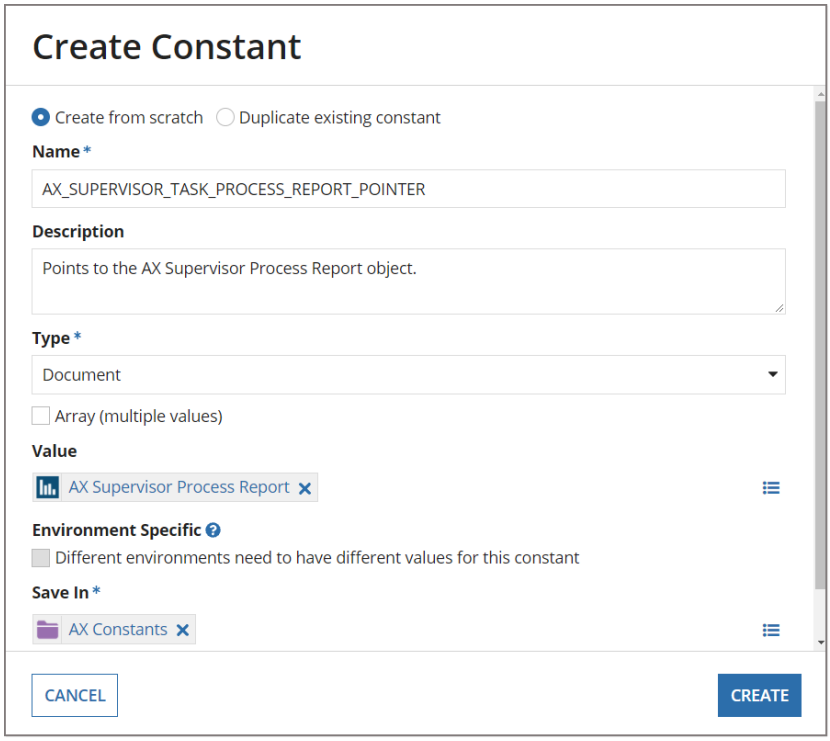


4. Click **Save** to to close the dialog box, and then click the save icon on the report menu bar to save the process report.

Create a Constant for the Process Report

Before you create the task report interface, you need to create a constant for the AX Tasks Process Report. This constant will be used in the interface that you will create in the next step.

1. From within the **AX Constants** folder, click **New**, and select **Constant** from the dropdown menu.
2. Type **AX_SUPERVISOR_TASK_PROCESS_REPORT_POINTER** for the **Name**, and add a description: "Points to the AX Supervisor Process Report object."
3. In the **Type** field, select **Document**.
4. In the **Value** field, type and select **AX Supervisor Process Report**. Click **Create**.



Create an Interface for the Task Report

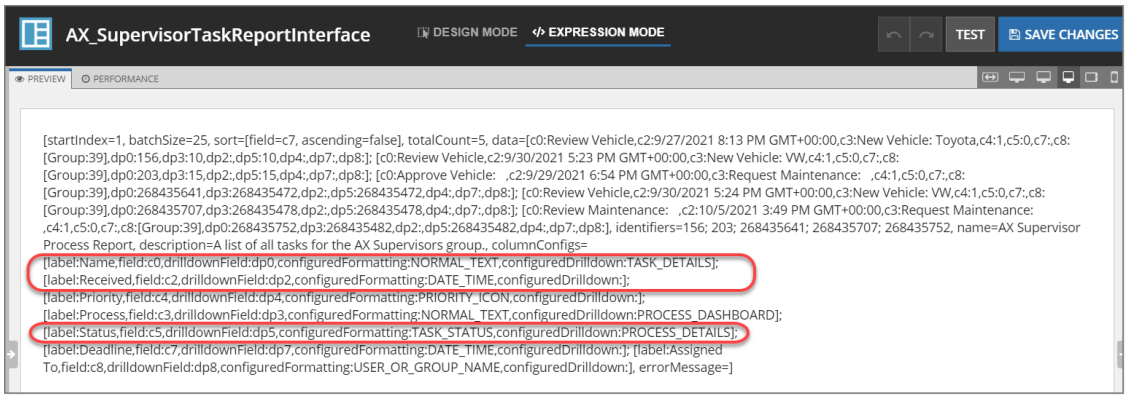
In this section, you'll create an interface to display all tasks in a grid. Later, you'll use this interface to create a task report.

1. From within the **AX Interfaces** folder, click **New**, and select **Interface** from the dropdown menu. Name it **AX_SupervisorTaskReportInterface**, and add a brief description "Interface to display tasks to Supervisors." Make sure the **Save In** folder is set to **AX Interfaces**, and click **Create**.

2. First, you will use the a!queryProcessAnalytics() function to populate the empty interface with data from the active tasks process report. You will do this in order to figure out what columns you want to display to Supervisors. In the interface, click **Expression Mode**, and enter the following expression into the **Interface Definition**:

```
a!textField(readOnly: true, value: a!queryProcessAnalytics(
report: cons!AX_SUPERVISOR_TASK_PROCESS_REPORT_POINTER ))
```

3. Scan the data in the **Preview** tab. You'll notice that it is the text representation of the process report data.



The columns you want to display in the grid are:

- C0 - Task name
- C2 - Date Time
- C5 - Task Status

4. Now that you know which columns you want to display, you are ready to set up your read-only grid. Delete the code in the **Interface Definition**, and then replace it with the following expression:

```
a!sectionLayout(
  contents: { a!gridField(
  label: "Supervisor Tasks",
  instructions: "A list of all tasks for the Supervisor
group",
  labelPosition: "ABOVE",
  data: a!queryProcessAnalytics(
    report: cons!AX_SUPERVISOR_TASK_PROCESS_REPORT_POINTER,
    query: a!query(pagingInfo: fv!pagingInfo)),
  columns: {
    a!gridColumn(
      label: "Name",
      sortField: "c0",
      value: fv!row.c0 ),
    a!gridColumn(
      label: "Status",
      sortField: "c5",
      value: fv!row.c5),
    a!gridColumn(
      label: "Date",
      sortField: "c2",
      value: fv!row.c2 ),},
  rowHeader: 1)})
```

5. Switch back to **Design Mode** and click **Save Changes**.

Create Links to Process Tasks

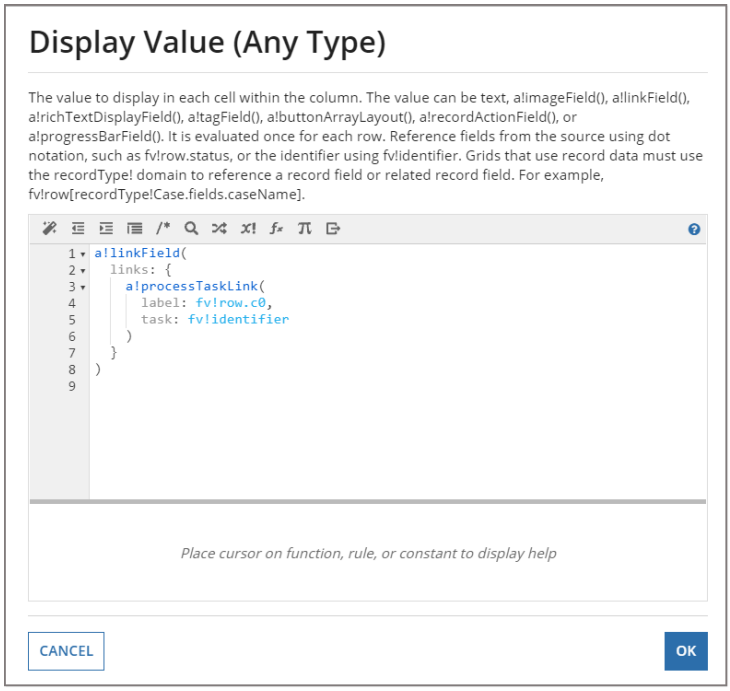
The grid you just created displays the task name but users do not have a way to open the task. In this section, you'll link the Name column to the actual task.

1. Select the **Read-only Grid** component, then under **Columns** in the **Component Configuration** panel, click **Name (Grid Column)**.

2. Click **Display Options**, then select **Process Task Link**.

3. Click the **Expression Editor** icon next to **Display Value**, and replace the existing expression with the following expression:

```
a!linkField(
  links: {
    a!processTaskLink(
      label: fv!row.c0,
      task: fv!identifier
    )
  }
)
```

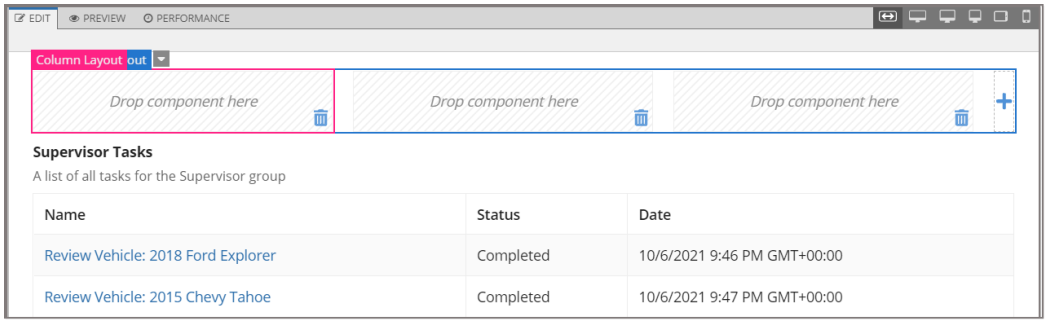


4. Click **OK**, then **Save Changes**.

Create the Status Filter

In this exercise, you'll format the status column to show meaningful values instead of numbers, and you'll add a filter to the interface to allow Supervisors to filter tasks by the most common statuses.

1. From the **Components** palette, grab a **Columns** layout, and drop it above the read-only grid.

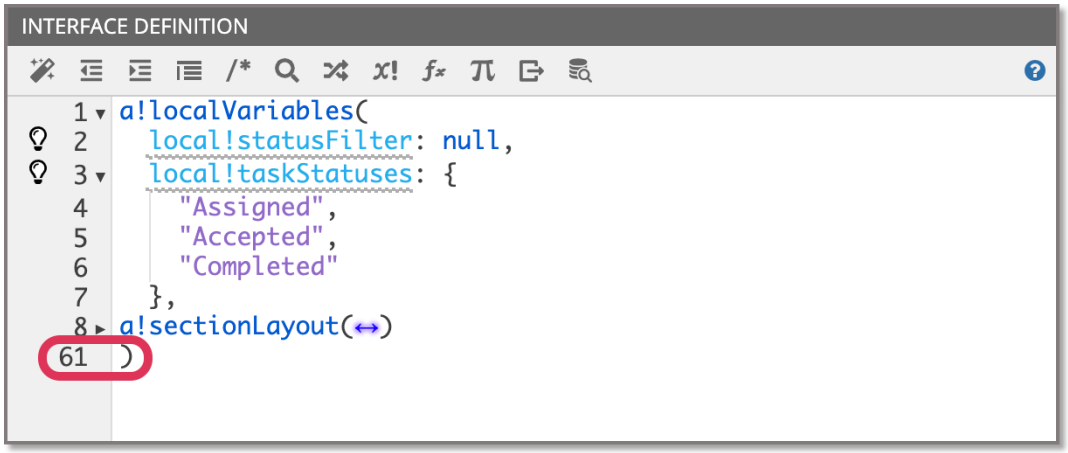


2. Next, let's add a local variable to this interface to hold the task status values that you will use in the dropdown component. This component will allow Supervisors to filter tasks by status. Click **Expression Mode**.

3. Click the arrow next to line 1 to collapse the existing expression, and press **Enter**. In line 1, type the following expression:

```
a!localVariables(
  local!statusFilter: null,
  local!taskStatuses: {
    "Assigned",
    "Accepted",
    "Completed"
  },
```

4. Scroll to the last line of code in the interface definition pane and add a right parenthesis ")". Now, our entire expression is encapsulated within the a!localVariables function. This is required so that local variables declared in the beginning can be referenced by other encapsulated code further down:



5. Return to the **Design Mode**, and click **Save Changes**.

6. Next, let's drag and drop a **Dropdown** field from the **Components** palette into the first column layout on the left.

7. In the **Component Configuration** pane, configure the dropdown field as follows:

    - Type **Status** in the Label field.
    - Click the **Expression Editor** icon next to **Choice Labels**, and type the following expression: **local!taskStatuses**. This populates the Choice Labels with the list of statuses defined in Step 4.
    - Click the **Expression Editor** icon next to **Choice Values**, and type the following expression: **enumerate(3)**.
    - In the **Selected Value** and **Save Selection To** fields, use the dropdown menu to select **local!statusFilter**.

8. Next, let's add a filter to the read-only grid, so that once the Supervisor selects a value in the Status dropdown, the grid refreshes to display only the selected statuses. Select the **Read-only Grid** component, and in the **Component Configuration** pane click the **queryProcessAnalytics** link (located in the **Data Source** section).

9. Click the **query** link in **ContextGroups**, and then click the **Edit** link below **filter**. Enter the following expression into the filter (QueryFilter) dialog box:

```
a!queryFilter(
  field: "c5",
  operator: "=",
  value: local!statusFilter,
  applyWhen: not(isnull(local!statusFilter))
)
```

    Click **OK**. Your filter should now work. Test the filter by selecting different values in the dropdown status menu.

10. Select the **Read-only Grid** component to replace the numbers in the Status column with labels like Assigned and Completed. In the **Component Configuration** pane, click **Status (Grid Column)** under **Columns**.

11. Click the **Expression Editor** icon next to **Display Value**, and replace the existing expression with the following expression:

```
a!forEach(
items: fv!row.c5,
  expression: if(
    not(isnull(fv!row.c5)),
```

```
fn!index(
  local!taskStatuses,tointeger(fv!item)+1,"Other"),
  tostring(fv!item))
```

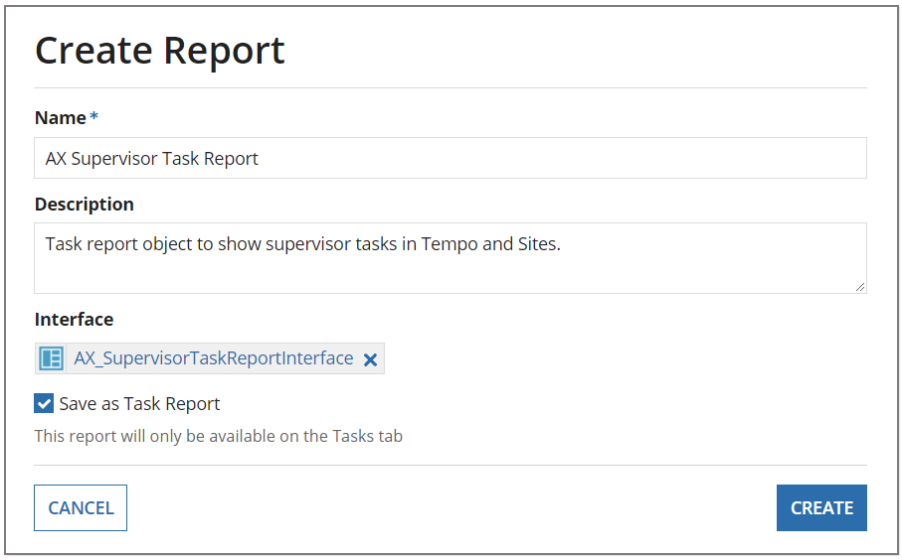
12. Click **OK**, then **Save Changes** to save the interface.

Create the Task Report

Now that you have the AX_SupervisorTaskReportInterface, let's create a task report object. Once you have a task report, you'll be able to add it as a page to the Acme Exercise Site.

1. In your application, click **New**, and select **Report**. Name this report **AX Supervisor Task Report**, and add a description: "Task report object to show Supervisor tasks."

2. Type and select **AX_SupervisorTaskReportInterface** in the **Interface** field, and select the **Save as Task Report** checkbox. Click **Create**.



3. To set security for this report, click **Add Users and Groups**, and add **AX All Users** as **Viewers**, and **AX Administrators** as Administrators. Click **Save**.

Add the Task Report Page to the AX Site

Next, add the AX Supervisor Task Report to the Acme Exercise Site.

1. Open the **Acme Exercise Site**, and click **Add Page**. Configure this page as follows:

    - Title the page **Supervisor Tasks**.
    - Click the dropdown menu for the icons, and then browse or search for an appropriate icon.
    - Select **Report** in the **Type** field.

- Select **AX Supervisor Task Report** in the **Content** field.
- Select **Only show when** under **Visibility**, and enter the expression:

```
a!isUserMemberOfGroup(loggedInUser(),
cons!AX_SUPERVISORS_POINTER)
```

- Click **OK**.

2. Click **Save Changes**. To preview the new Supervisor Tasks page, click the **URL** for the site, and select the new Supervisors Tasks tab. Now, the Supervisors are able to see their tasks in a single place, filter them by status, and approve or deny the addition of vehicles to the fleet.

Export the Application

The import and export of applications is disabled for Appian Community Edition. However, for applications outside of Appian Community Edition, you will want to export your finished application as a zip file. Once you export the app, you can store the file, or import it into a different environment. In order to export the app, you will prepare for the export by:

- Reviewing your app's security settings
- Checking for missing precedents, or objects

These two quick checks will ensure that your app is export-ready and will work as expected in its new environment. When you export future applications, follow the steps below.

Review Security Summary

1. Check the security summary of the app to ensure that every object has been secured correctly. Click the **Settings** menu  at the top right of the app, and select **Security Summary** from the dropdown menu.

2. All objects should have a security setting, and only groups and not individual users should be used for security. Review your **Security Summary**, making sure that there are no unsecured objects, and all objects are secured at least through the **AX Administrators** group. Review that the **AX All Users** group is assigned the **Viewer** rights.

Check for Missing Precedents

A missing precedent refers to an object referenced in your application that belongs to a different app. Usually, this happens when you reuse a shared object. To prevent problems, always check for missing precedents, and add the shared objects to your app before exporting it. Follow the steps below to complete this important check.

1. Click the **Settings** menu  at the top right of the app, and select **Missing Precedents** from the dropdown menu.

2. If there are any missing precedents in your app, review them to verify whether you need to add the objects to your app, or the dependency should be removed. For example, sometimes developers may include an erroneous object into their app by clicking the incorrect auto-suggestion. There are other reasons why you may get a missing precedent message:

    - Many teams create a dedicated application for deploying shared components. This is done to avoid clutter. If this is your scenario, you may need to ignore the missing precedent message. The app with shared components is usually deployed separately, or it might be already present in your target environment.
    - Missing precedents can also occur if you erroneously create an object from within a different app. In this instance, add the object to your app.

3. If you find a precedent that should be added to your app, select the precedent and click **Add to App**. Alternatively, you can add multiple precedents by clicking **Add all to application**. Click **Check Again** to run another missing precedents check that no further dependencies were discovered.

Export the Application

Once you complete the above checks, export your app.

1. Click the **Settings** menu  at the top right of the app, and select **Export Application** from the dropdown menu. Verify the name of your app, and update it to include the date of the export.

2. Click **Export**, and then click **Download Package**.

3. If you receive an error message, open the export log and correct any problems. Download the newly created **zip file**, so you can take your work with you to a different environment.