

Form Validations

Exercise - How to

Outline	2
How to	2
MovieDetail Screen Validations	2
PersonDetail Screen Validations	9

Outline

In this exercise, we will focus on adding some business logic to our app as we validate the forms we have already created. This will be done in the MovieDetail and PersonDetail Screens to avoid saving invalid information in the database.

At the end of this exercise, we want to guarantee that the following rules are followed:

- A movie's gross takings can never be a negative number.
- A movie that wasn't released yet cannot have any gross takings amount.
- A person's date of birth cannot be in the future.
- A person's date of birth cannot be later than the person's date of death.

If a movie or person was successfully added to the database, then a feedback message should be displayed on the screen.

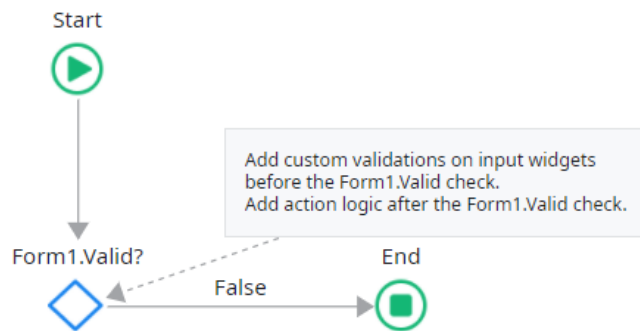
How to

In this section, we'll describe exercise 6 – *Form Validations*, step by step.

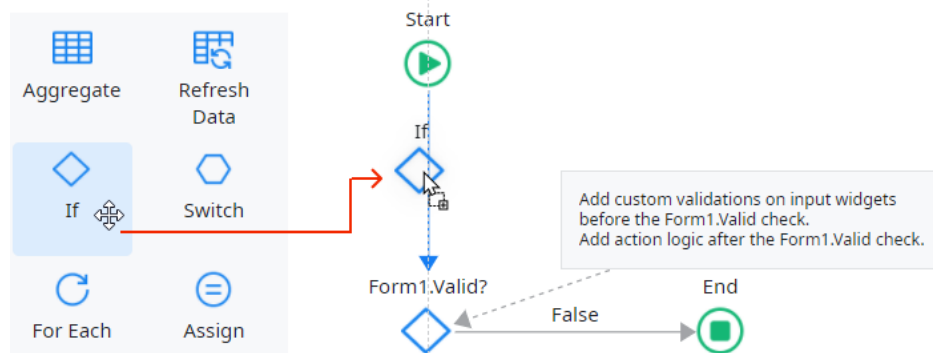
MovieDetail Screen Validations

We will start this exercise by focusing on the movie validations: a movie's gross takings amount cannot be a negative number, and a movie cannot have a gross takings amount number if it hasn't been released yet. Let's work on the MovieDetail Screen to build these validations.

1. In the **SaveOnClick** Action of the MovieDetail Screen, create the logic to ensure that a movie's gross takings are never a negative number. To do that we need to work on the SaveOnClick Action and with the property of the gross takings amount input widget. If the data is not valid, a message should appear to the end-user, next to the gross takings amount input.
 - a. Open the **SaveOnClick** Client Action from the MovieDetail Screen.



- b. Drag an **If** to the Action flow and drop it before the existing If. You can also delete the gray box with the comment.

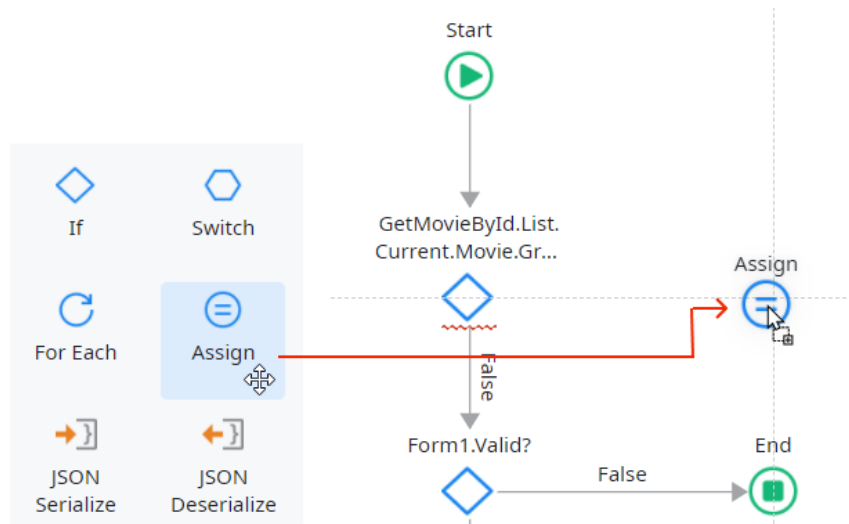


- c. Set the **Condition** of the If to be:

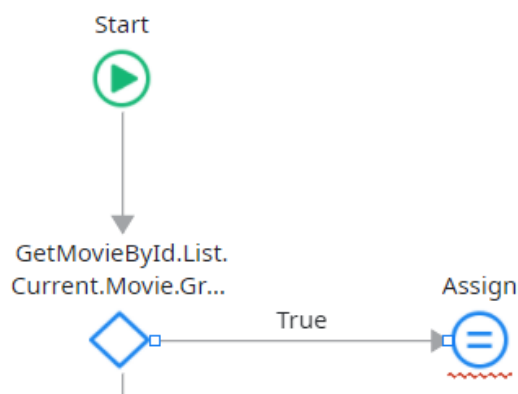
GetMovieById.List.Current.Movie.GrossTakingsAmount < 0

So, we're using an If to check if the gross takings amount value of the movie is smaller than zero. Here, we are leveraging the output of the Aggregate, since it's where the data about the movie comes from.

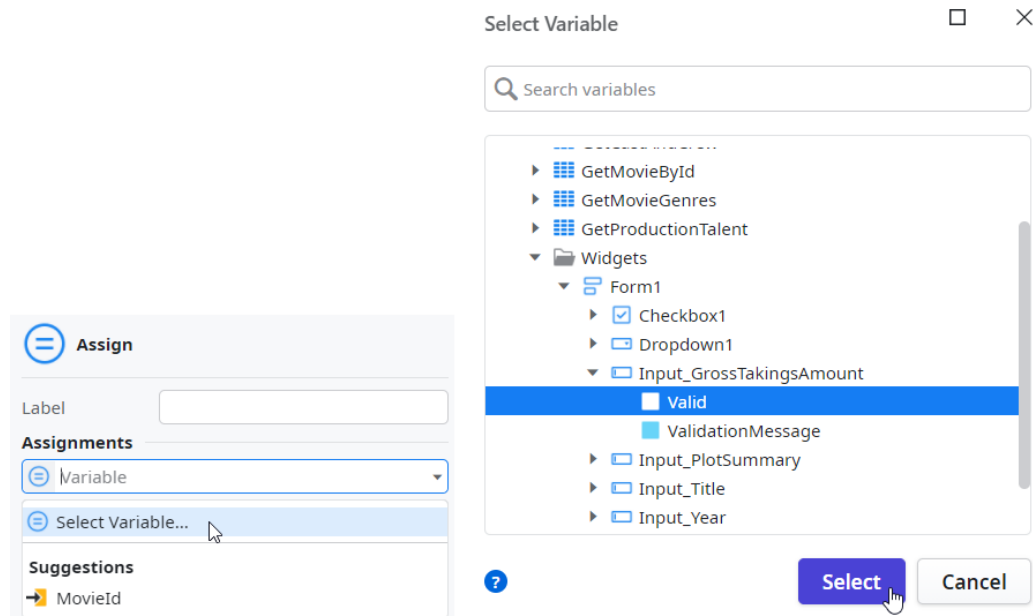
- d. Drag an **Assign** and drop it next to the recently added If.



- e. Connect the If to the Assign node. This creates the **True** branch, indicating that if the Condition is true, the input data is not valid, since the gross takings amount is a negative number.

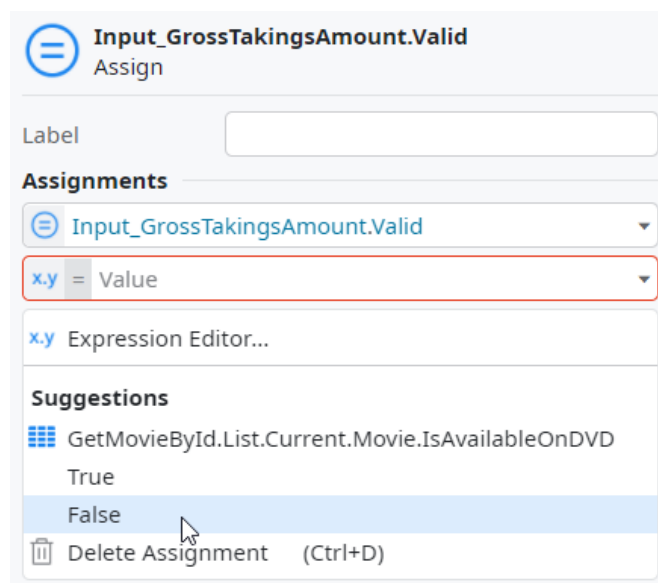


- f. In the Assign properties area, click on the **Variable** property and choose **Select Variable**. Then, expand the Widgets folder, expand the Form, and select the **Valid** property of the **GrossTakingsAmount** input field.



Note: Each input field has a Boolean property called **Valid** that can be programmatically manipulated to validate / invalidate an input field. In this case, the Action flow will get to this Assign when the Gross Takings Amount of a movie is a negative number. In this Assign we can set the input field as invalid, to let the user know that the value being inserted is not valid. How does that happen? The input field will, by default, have a red border surrounding it.

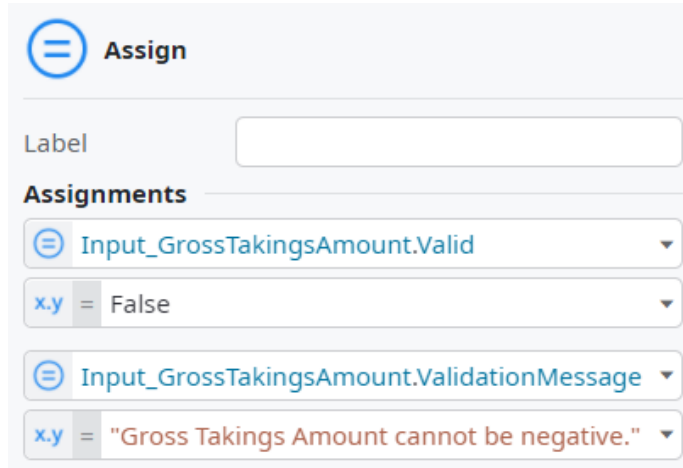
- g. So, how to invalidate the input field? Set the **Value** for the Assign to be *False*.



This alone invalidates the input field. But we want to have a message appearing next to the input field, whenever the input field is not valid.

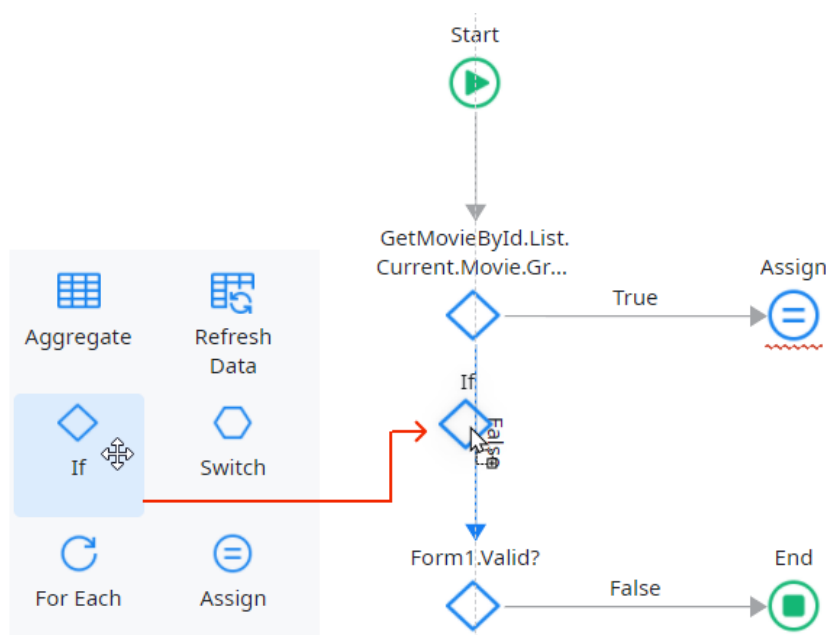
- h. In the same **Assign** node, create the following assignment after the previous one to display an invalid input message to the end-user:

Input_GrossTakingsAmount.ValidationMessage = "Gross Takings Amount cannot be negative."



Note: The ValidationMessage property is often used together with the Valid property, since it sets the message that will appear to the end-user. Every input widget has these two properties.

2. In the same Action, create the logic to validate that a movie cannot have any gross takings, if it wasn't released yet.
- a. Drag another **If** and drop it between the two existing ones.



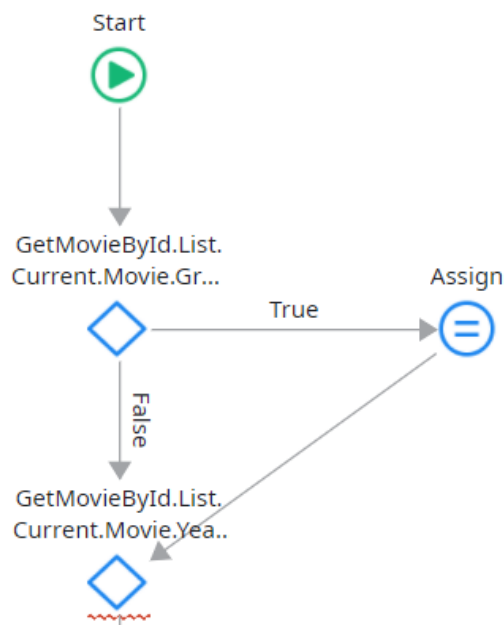
- b. Set its **Condition** to:

*GetMovieById.List.Current.Movie.Year > Year(CurrDate()) and
GetMovieById.List.Current.Movie.GrossTakingsAmount > 0*

Note: The *CurrDate* function returns the current date, while the *Year* function returns the year of a date.

This condition checks if a movie's release date (year) is bigger than the year we are on.

- c. Connect the existing **Assign** to the new If.

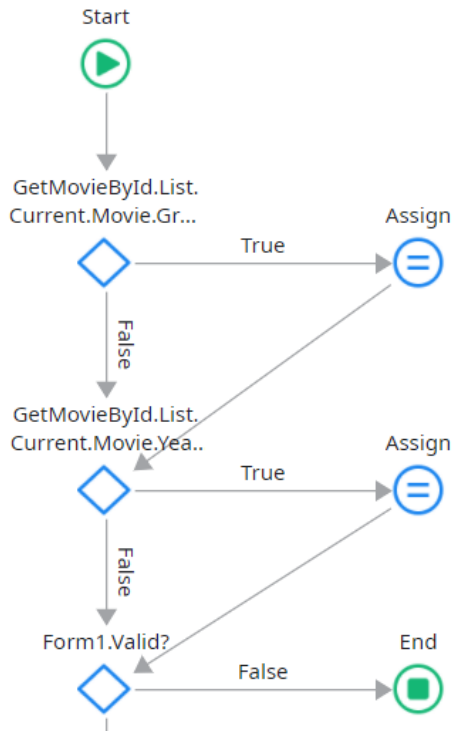


- d. Drag a new **Assign** and drop it to the right of the recently added If node. Connect the two nodes together and define the following assignments.

*Input_GrossTakingsAmount.Valid = False
Input_GrossTakingsAmount.ValidationMessage = "The movie cannot have any
gross takings since it was not released yet."*

This is following the same strategy used in the first validation.

- e. Connect the new Assign to the **Form1.Valid?** If.



Note: So, why is the Form1.Valid If there? That's the final gatekeeper for our validations. If one of the input fields inside the Form is not valid, then the Form automatically becomes invalid as well. If you notice, the logic flow can reach that If through multiple paths, so that's the final validation that we do. If the logic follows the "happy path" and all the validations are checked successfully, then the Form will be valid and the logic can proceed. Otherwise, the form is not valid, and we cannot add or update the movie in the database.

- f. Publish the module and test the validations in the browser by inputting valid and invalid data.



You should see the messages appearing when the data is not valid.

Gross Takings

-1

Gross Takings Amount cannot be negative.

Is Available On DVD



[Back to Movies](#)

Save

PersonDetail Screen Validations

Now, we will move to the PersonDetail Screen and implement the other two validations.

1. In the **SaveOnClick** Action of the PersonDetail Screen, create the validation logic to guarantee that the person's birth date is not in the future.
 - a. Open the **SaveOnClick** Action of the PersonDetail Screen.
 - b. Drag an **If** to the Action flow and drop it below the Start node and set its **Condition** to:

GetPersonById.List.Current.Person.DateOfBirth > CurrDate()

- c. Drag an **Assign** node and drop it next to the If. Don't forget to connect the If to the Assign, creating the true branch.
 - d. Set the following assignments:

Assign

Properties Styles

Label

Assignments

Input_DateOfBirth.Valid

x.y = False

Input_DateOfBirth.ValidationMessage

x.y = "Date of birth cannot be in the future"

Variable

x.y = Value

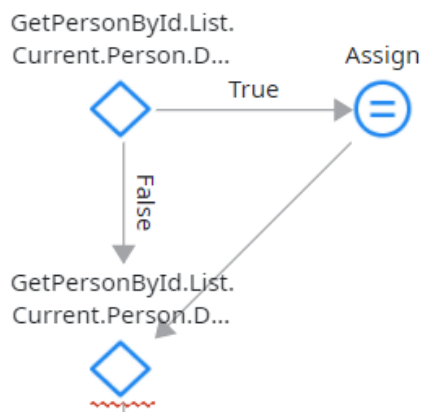
2. Now, create the validation logic to guarantee that the person's date of death cannot occur before the date of birth.

- a. Drag another **If** between the two existing ifs and set the **Condition** to:

*GetPersonById.List.Current.Person.DateOfDeath <> NullDate() and
GetPersonById.List.Current.Person.DateOfDeath <
GetPersonById.List.Current.Person.DateOfBirth*

This condition is a bit trickier. The second part, after the And, is straightforward. We're checking if the date of death is smaller than the date of birth. Now, the first part of the condition may not be easy to understand at first. There's one thing that is very important to remember: the date of death is not a mandatory field in the database, so it can be empty. So, before checking if the date of death is smaller than the date of birth, we need to check if the date of death has indeed a value. If it doesn't, then there's no need to even check the second part. If it does, then it has to be bigger than the date of birth.

- b. Connect the **Assign** for the Date Of Birth in the future with the recently created If.



- c. Drag a new **Assign** node next to the If and set the following assignments. Don't forget to connect the new If with this new Assign.

Assign

Properties Styles

Label

Assignments

- d. Connect the Assign with the **Form1.Valid?** If.
- e. Publish the module and test the new validations with invalid and valid values.



All right! We have some validations up and running. As a final note for this exercise, all these validations are being performed in Client Actions. In general, the client-side of any app is more susceptible to attacks, so it's a best practice to also protect our data on the server-side, where it's considerably more difficult to attack. However, we have made an excellent progress so far, and we will leave it as it is.