outsystems

# Pagination and Sorting

## Exercise - How to

# Outline

In this exercise, we will add pagination and sorting functionality to the Movies Screen of the OSMDb application.

Our application can have multiple movies, which can lead to the Table in the Movies Screen to have a lot of records to display. To have a better user experience, we want to add the pagination functionality, so that each page only shows five movies at a time.

Then, we want to enable the users to be able to dynamically sort the movies, by clicking on any of the column headers in the Table. This means that if the user clicks on the Title header, the movies should appear sorted by movie title.

At the end, we have an extra challenge to implement two scenarios:

- When the user clicks on a column header twice in a row, the order of the sorting should change from ascending to descending.
- When the sorting by a column is done, the pagination should reset to the first page.

# How to

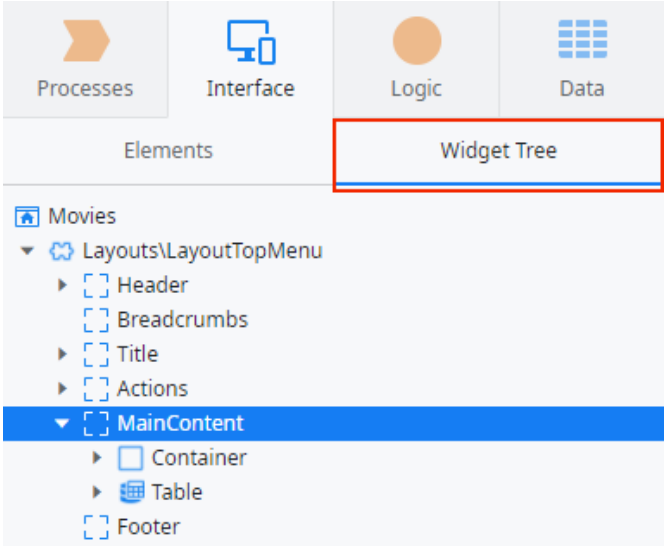
In this section, we'll describe, step by step, the exercise *10 - Pagination and Sorting*.

## Pagination

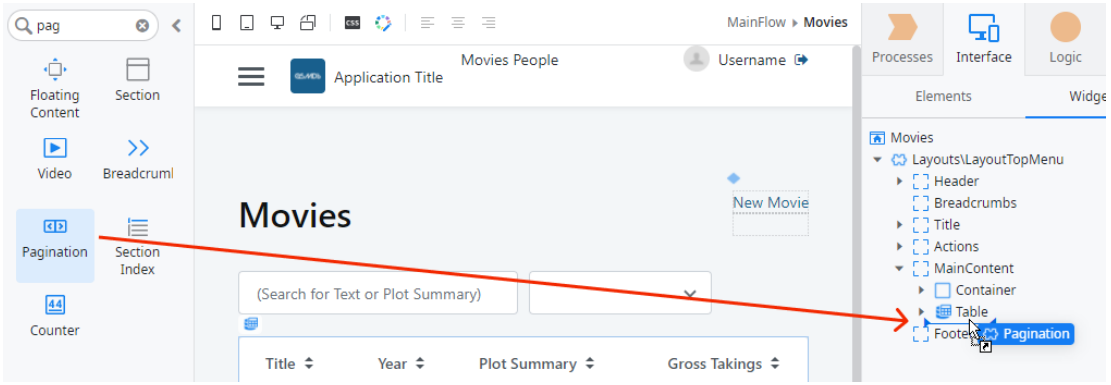
First, we will implement a pagination functionality on the Movies Table, using a Pagination widget. With that we want to display just five movies per page.

1. Add the Pagination widget to the Movies Screen, right after the Movies table. Create 2 Local Variables for representing the Start Index of each page and the maximum number of movies that we want to display per page (MaxRecords).

a. Open the **Widget Tree** on the Movies Screen and expand the MainContent.




b. In the widget toolbar on the left, type *pag*, find the **Pagination** widget, drag it and drop it after the Table.

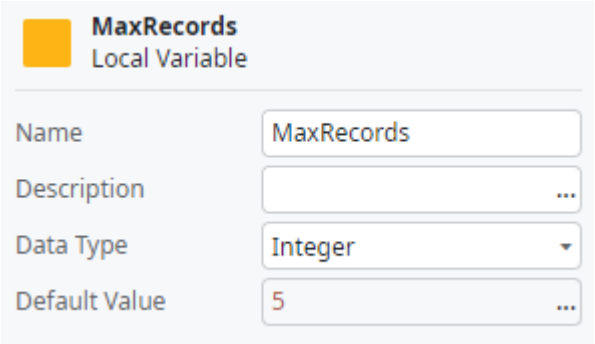



---

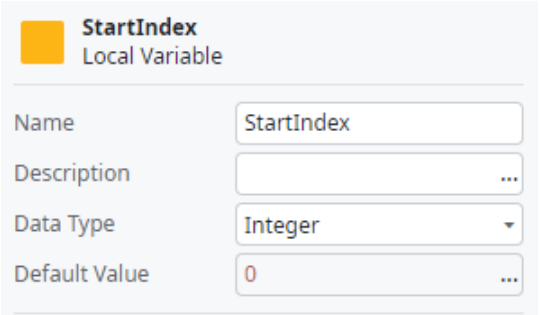
**NOTE:** The Pagination has four properties that have an error at this point: *StartIndex*, which represents the first record in each page; *MaxRecords*, which represents the number of records in each page; *TotalCount*, which represents the total number of records displayed on the Table; Event Handler, which expects an Action that will be triggered whenever the user changes page.

---

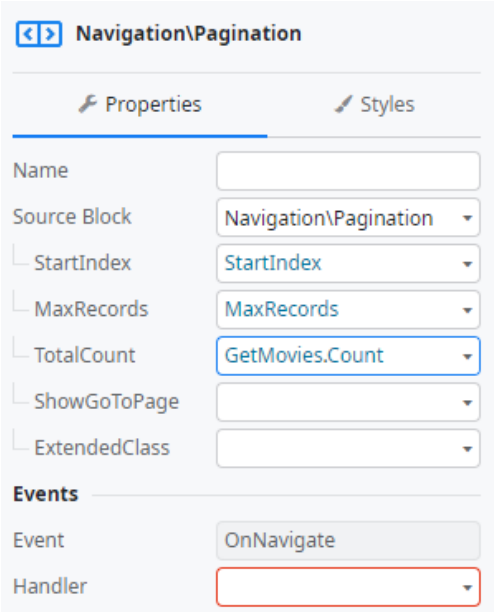
c. Create a local variable *MaxRecords*. It's **Data Type** should be set to *Integer* and the **Default Value** should be set to *5*




d. Create a Local Variable *StartIndex*. It's **Data Type** should be set to *Integer* and the **Default Value** should be set to *0*

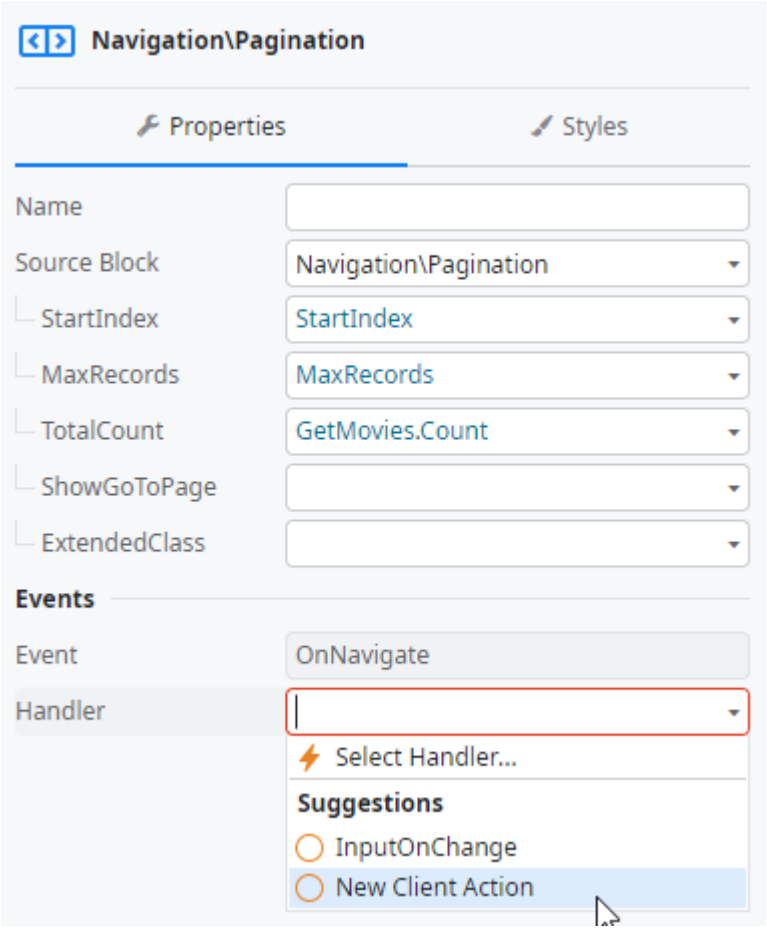



e. Select the Pagination widget and set **StartIndex** and **MaxRecords** properties to the respective Local Variables. Also, set the **TotalCount** to *GetMovies.Count*, which gives us the number of records returned by the GetMovies Aggregate.



2. Create the logic to support the behavior when a user changes the page. The Pagination has a Handler that should trigger a Client Action. Within that Action, we should implement the logic to change the Start Index and fetch the records for that page.

   a. In the same property dialog, expand the **Handler** property and choose *(New Client Action)*. This Action will have the logic that supports the change of page.




   b. A new Action, *PaginationOnNavigate*, is created with an Input Parameter: *NewStartIndex*. This Input will pass to the Action the information about the Start Index of the page clicked.

   c. Drag an Assign and drop it on the Action flow. Set the assignment to be:

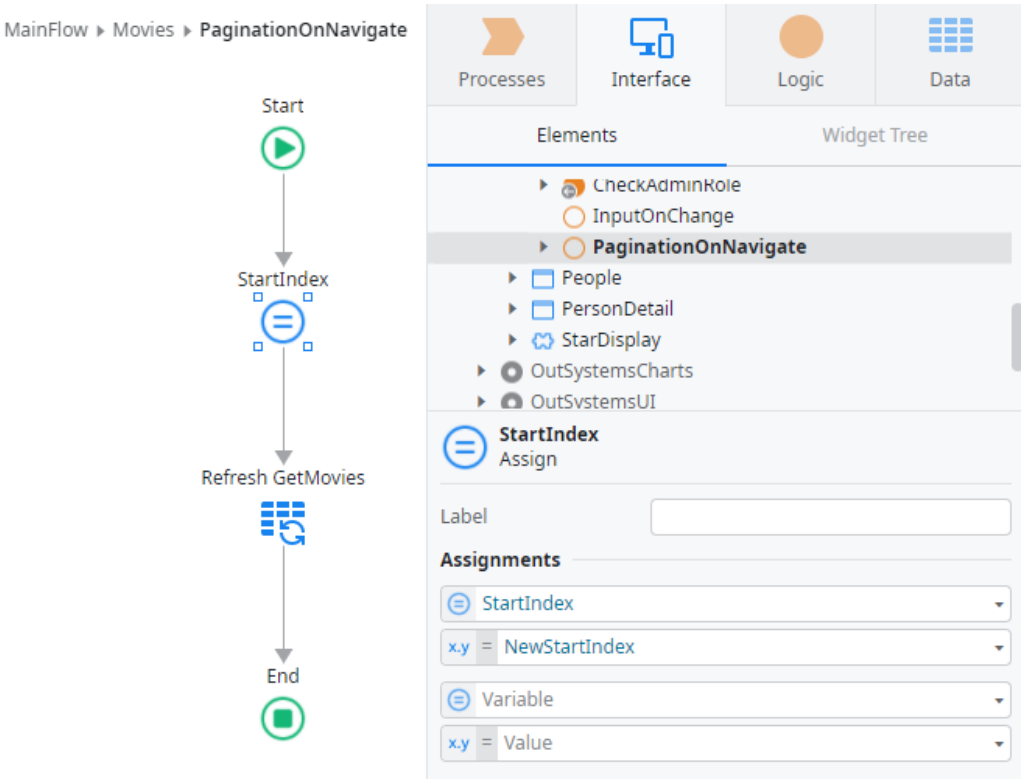
      *StartIndex* = *NewStartIndex*

---

**NOTE:** This assignment stores the value of the new start index, given by the page selected by the user, in the StartIndex Local Variables.

---

d. Drag a **Refresh Data** node and drop it on the Action flow. In the new dialog, select the **GetMovies** Aggregate. The Action flow should look like this:

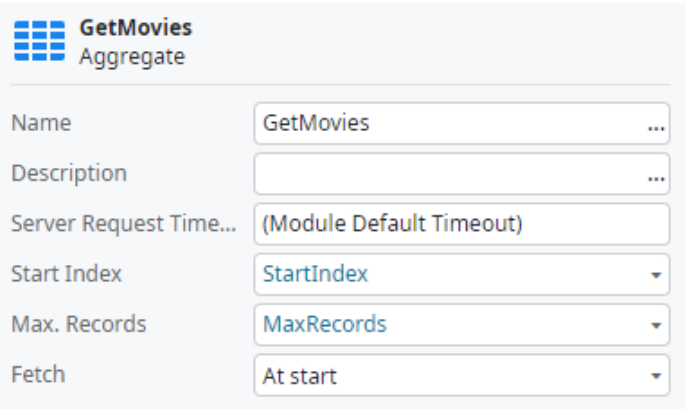



---

**NOTE:** The Refresh Data node will trigger the execution of the GetMovies Aggregate again. The idea is to get the records that correspond to the page that was selected. However, refreshing the Aggregate is not enough. To make this work, we need to define the Aggregate to only return the number of records per page (MaxRecords), starting from a particular record instead of always from the beginning (StartIndex).

---

3. To finalize, we need to change the **GetMovies** Aggregate to consider the StartIndex and the MaxRecords values. Just set the properties to the respective Local Variables.



4. Publish the module and test the pagination in the Movies Screen. Make sure that the records change accordingly.

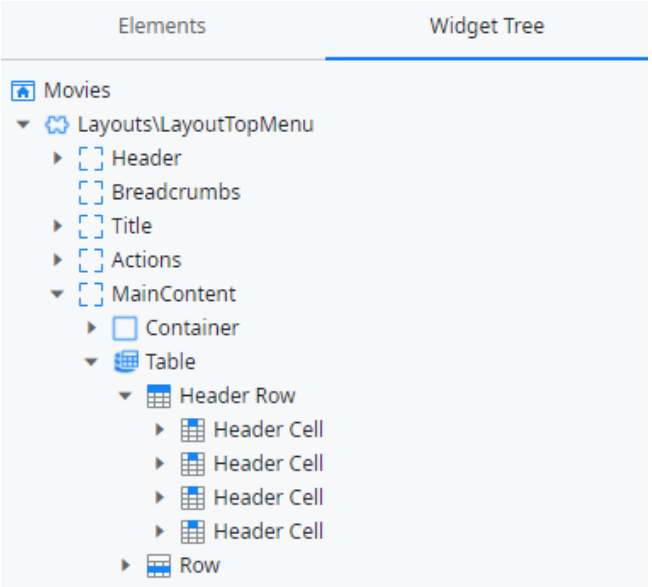
## Sorting

In the second part of this exercise, we will implement dynamic sorting on the Movies Table. Basically, whenever a user clicks on the header of a column of the Table, the results will automatically appear sorted by the values in that column.

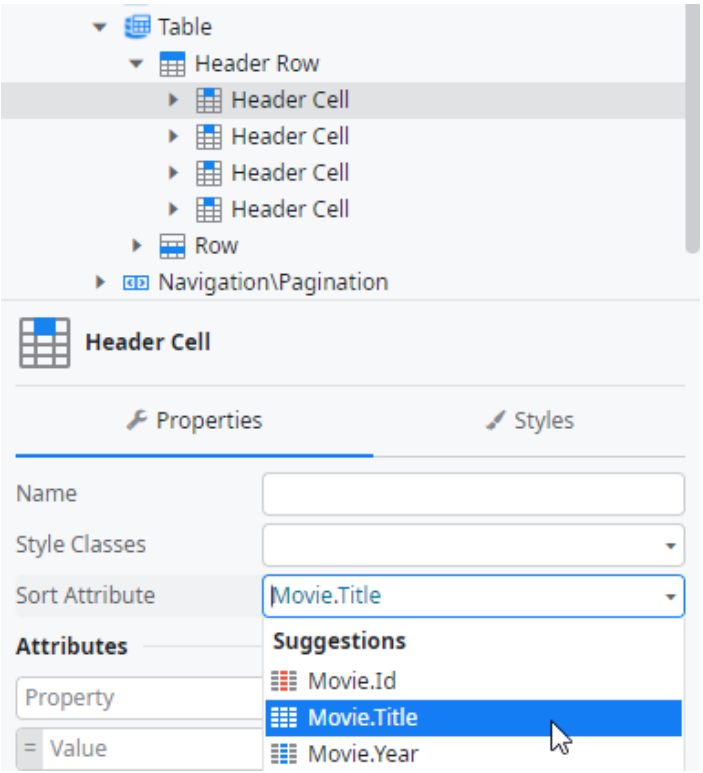
1. In the Movies Screen, define the Sort Attribute of each Header Cell, to the respective Entity attribute.

    a. Open the Widget Tree on the Movies Screen.

    b. Expand the MainContent, then the Table and finally the Header Row, until you see the Header Cells.

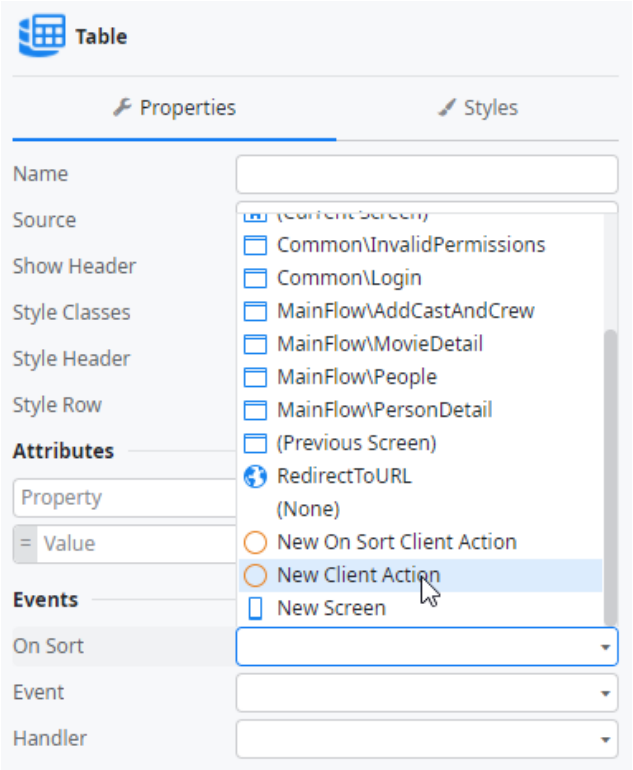


c. For each Cell, set the **Sort Attribute** property to the respective Entity attribute. For instance, the first Header Cell corresponds to the Title, so we should set the **Sort Attribute** to *Movie.Title*.

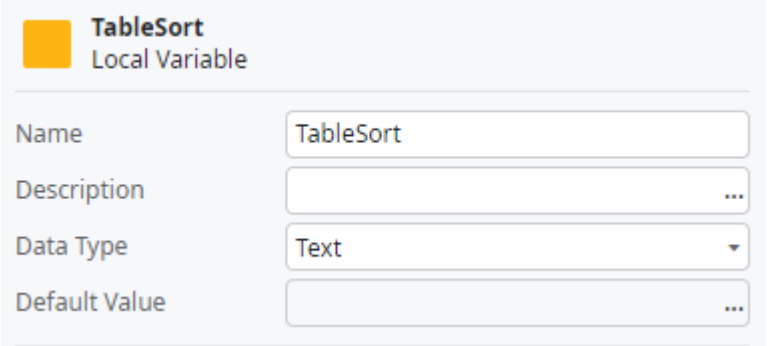



2. Set the On Sort Event of the Table to a new Client Action, where the logic for the sort will be implemented. This logic will require the sorting criteria to be stored in a local variable. Then, the start index should be reset and the Aggregate refreshed.

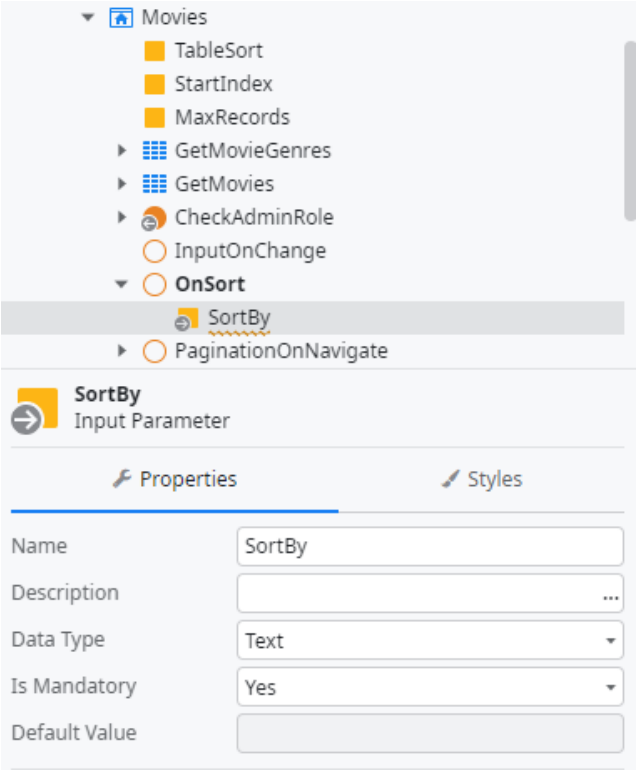
a. Select the Table and in the **On Sort Event** choose *(New Client Action).* This will create an *OnSort* Action.




b. Create a Local Variable on the Screen called *TableSort*, with **Data Type** set to *Text*.



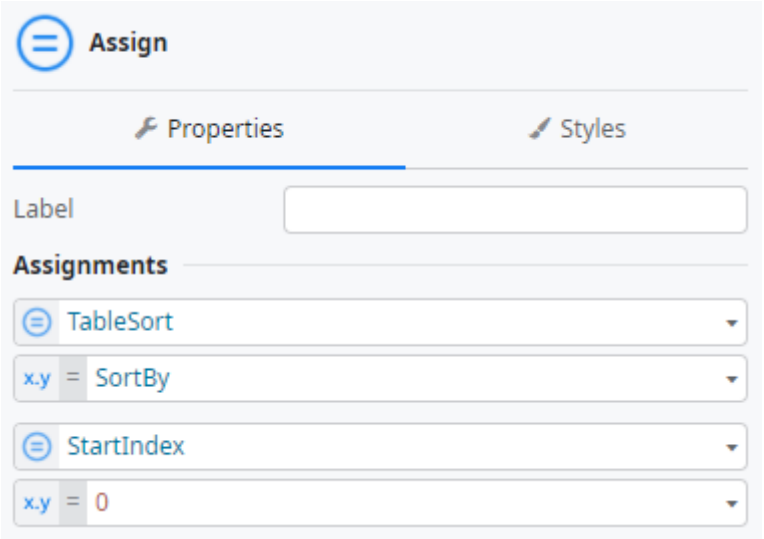
c. Add a new Input Parameter to the **OnSort** Action called *SortBy* with **Data Type** set to *Text*.




d. Drag an **Assign** and drop it on the OnSort Action flow. Define the following assignments:

*TableSort = SortBy*

*StartIndex = 0*

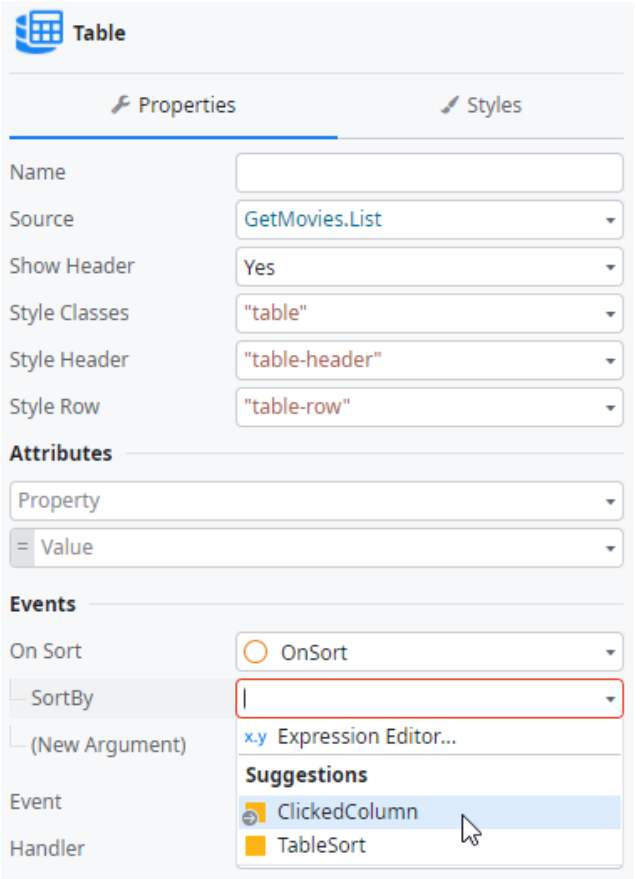


**NOTE:** The SortBy attribute will hold the value of the column selected by the user to sort by. So, we set the TableSort variable to that value passed as the input parameter of the OnSort Action. We will come back to this later.

The StartIndex is set to 0, so that after sorting, we come back to the first page of the Movies Table pagination.

e. Drag a **Refresh Data** node and drop it after the Assign. Select the **GetMovies** Aggregate in the new dialog.

f. To finish the logic, go back to the Movies Screen, select the Table and notice there is an error on the input parameter of the OnSort Action. Set the value of the input parameter to *ClickedColumn*.

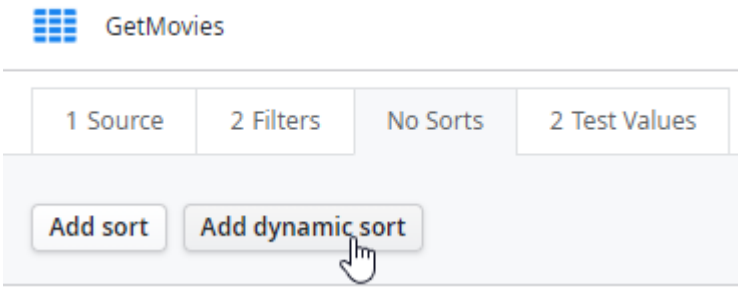



**NOTE:** The ClickedColumn input appears automatically when we are defining the On Sort Event Action handler. This ClickedColumn value will automatically hold the selection that the end-user makes during runtime, meaning the Sort Attribute of the column selected. Then, in the Action we save this information in the TableSort Local Variable, since we will use it later.

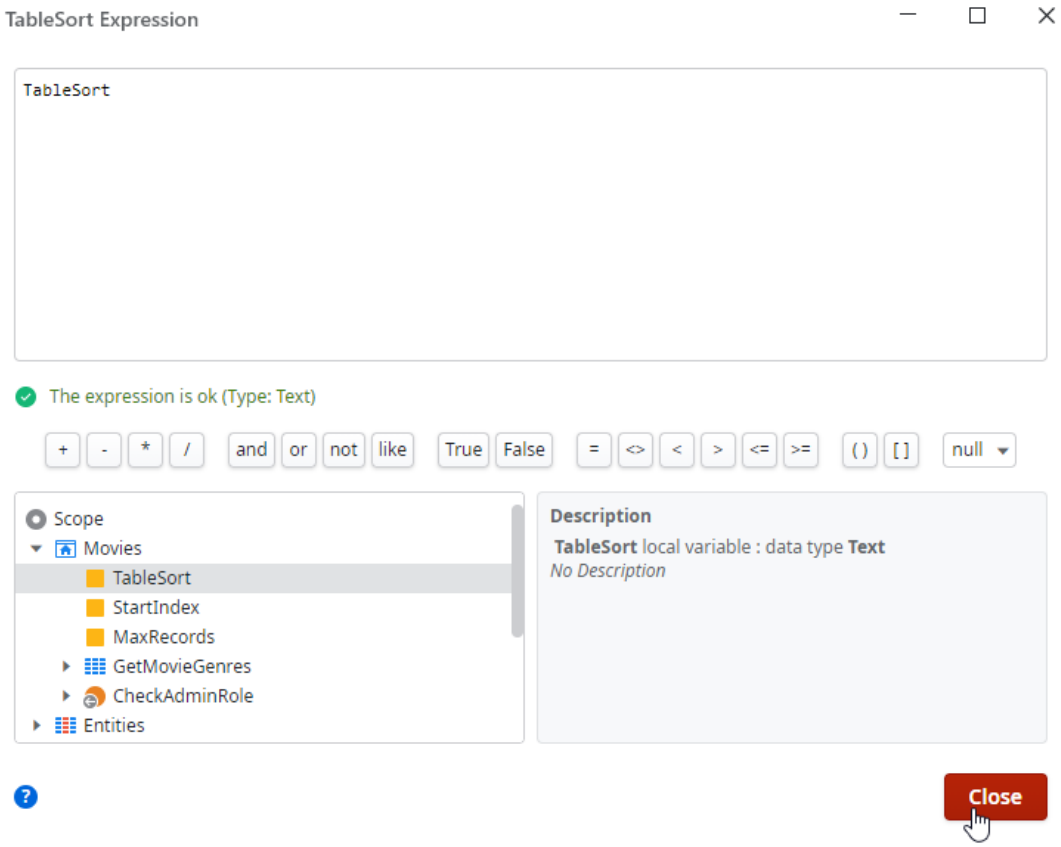
3. At this point, all the logic is set, but there is an important step missing. The Aggregate needs to be adjusted so that it fetches the data according to the sorting criteria selected by the user. To support that, we need to add a dynamic sorting to the Aggregate.

   a. Open the **GetMovies** Aggregate and select the **Sorting** tab.

   b. Click on the option **Add Dynamic Sort**




   c. Set the Expression to *TableSort.* Click **Done** to confirm.




**NOTE:** The TableSort Local Variable has the sorting criteria selected by the user, so by setting the dynamic sort to the value of the local variable, the Aggregate will be sorted by the selected criteria. The Add Sort option could not be chosen in this

case since it just implements static sorting. The Add Dynamic Sort is the option to go when implementing dynamic sorting on a Table / List.

---

4. Publish the module and test the application in the browser.