Sidebar Exercise

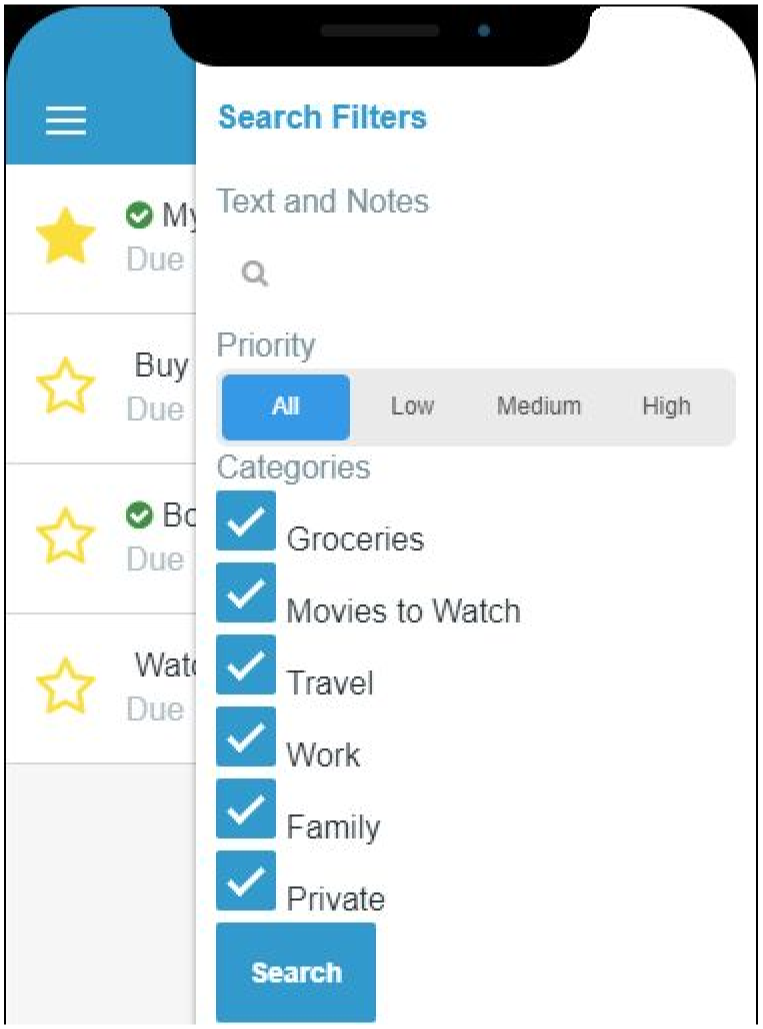


Table of Contents

Introduction

In this exercise lab, we will use the Sidebar Pattern from the OutSystems UI Framework. This Sidebar will be available in the ToDos Screen and will be used for filtering the list of To Dos displayed on the page.

First, we will create the Sidebar and make sure that it can open / close on click, and also by swiping left and right.

Then, we will start creating our input filters. We will start by allowing a search for Title and Notes, meaning that a user can type a keyword and any ToDo with that keyword in these two attributes will be displayed in the list of To Dos.

The next search filters we will add is by priority and categories, using ButtonGroups and Checkboxes.

For categories, this raises an issue with synchronization. Since the synchronization process deletes all the local categories before adding the information coming from the server, the search filter will be reset and all the Categories will be selected. To solve this, we need to fix the synchronization of categories to make sure that the search filter is kept after the synchronization process finishes.

Finally, we will add a Clear Filters Button to reset the search.

As a summary, in this specific exercise lab, we will:

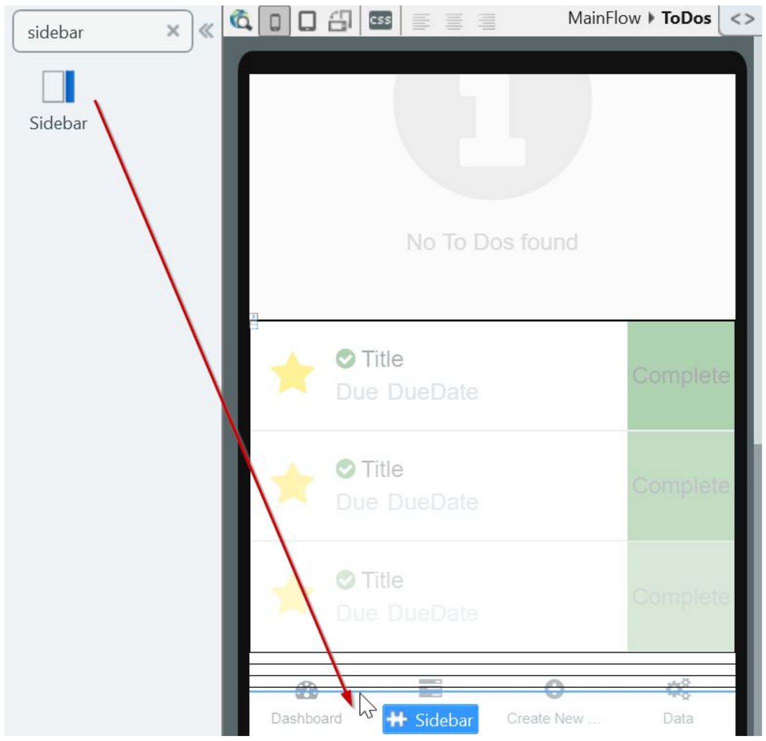
- Add a Sidebar to the ToDos Screen for filtering ToDos
- Define searches by Text, Notes, Priority and Category
- Tweak the synchronization of the categories to make sure that the search filter is not cleared after sync
- Add a Button to reset the filters

Creating a Sidebar to search for ToDos

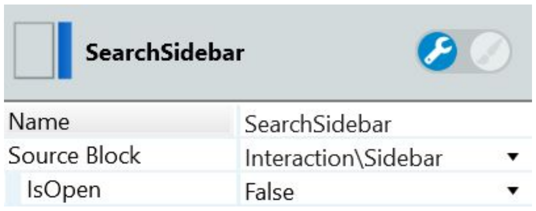
In this section of the exercise, we will add a Sidebar to the **ToDos** Screen. This Sidebar will enable users to search the existing To Dos based on the Title or Notes.

1) Add a **Sidebar** pattern to the **ToDos** Screen, by dropping it right after the List of ToDos. Define a Link on the **Header** of the Sidebar to close it when clicked. The Link should have an icon and the text *Search Filters*, with *heading4* Style. **Hint:** Use the **ToggleSidebar** Action to help closing / opening the sidebar.

    a) Open the **ToDos** Screen, and from the widget toolbar on the left, drag the **Sidebar** pattern and drop it on the end of the Screen.



    b) Set the **IsOpen** property of the Sidebar to *False* and the **Name** to *SearchSidebar*.

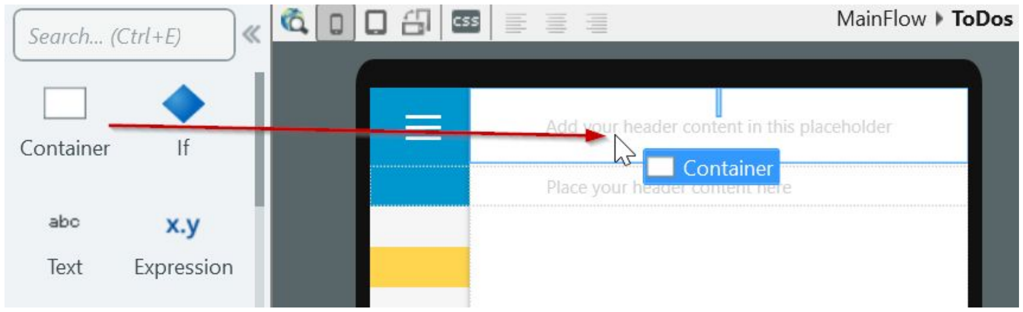


SearchSidebar

Name	SearchSidebar
Source Block	Interaction\Sidebar
IsOpen	False

NOTE: The **IsOpen** property helps handling the state (open or closed) of the Sidebar. The initial state of the Sidebar is set to *False*, meaning that it is closed.

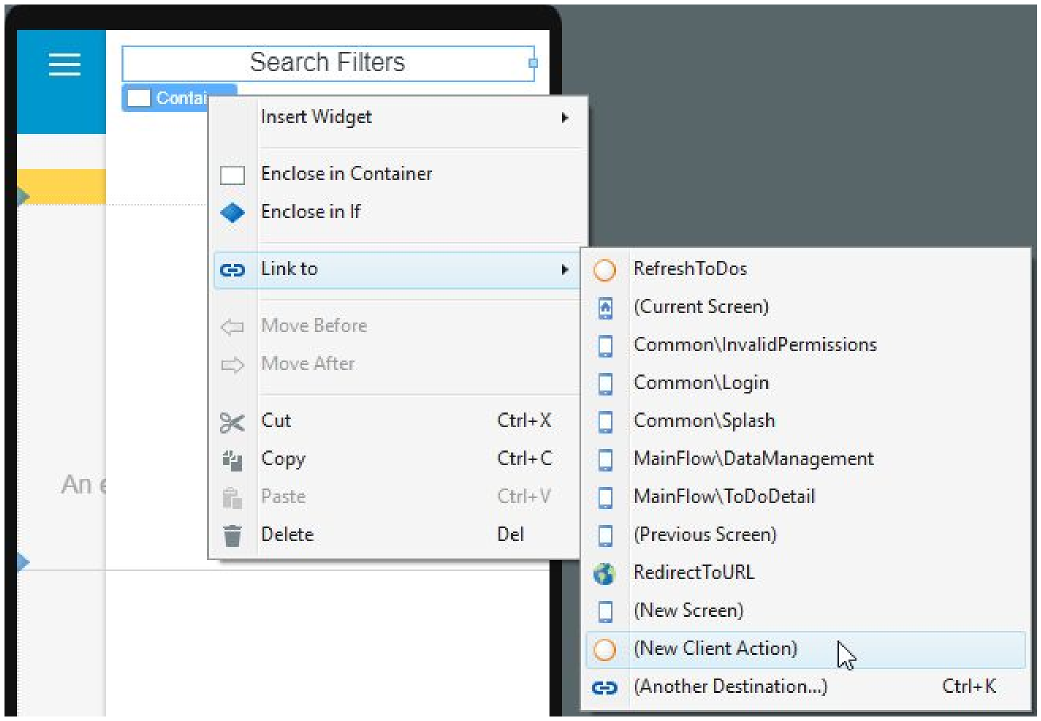
c) Drag a Container and drop it inside the **Header** placeholder of the Sidebar.


Search... (Ctrl+E)
Container
If
Text
Expression
MainFlow ToDos
Add your header content in this placeholder
Container
Place your header content here

d) Set the Style Classes property to *"heading4"*

e) Place the cursor inside the Container and type *Search Filters*

f) Right-click on the surrounding Container and choose *Link to* > *(New Client Action)*

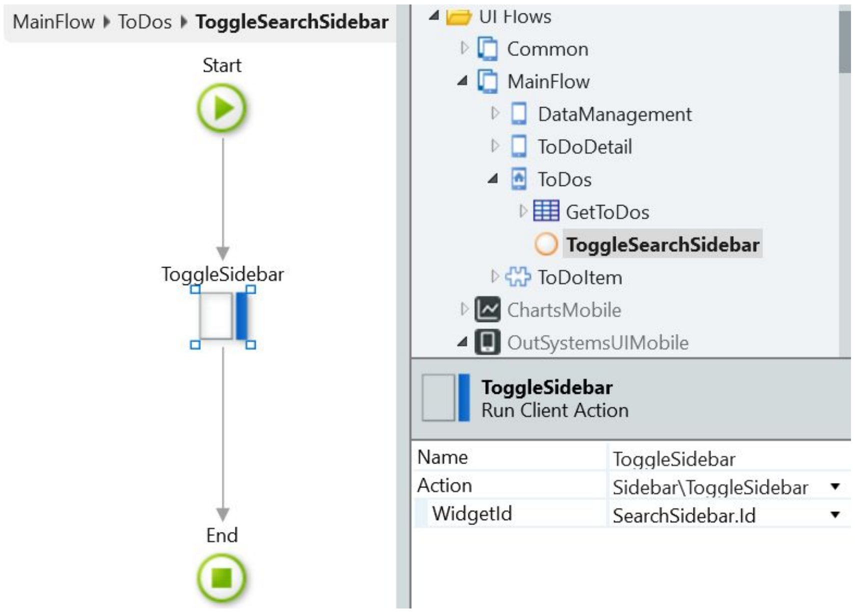


g) Set the **Name** of the new Action as *ToggleSearchSidebar*

h) Drag and drop a **Run Client Action** statement to the Action flow.

i) Select the **ToggleSidebar** Action. Set the **WidgetId** parameter to *SearchSidebar.Id*

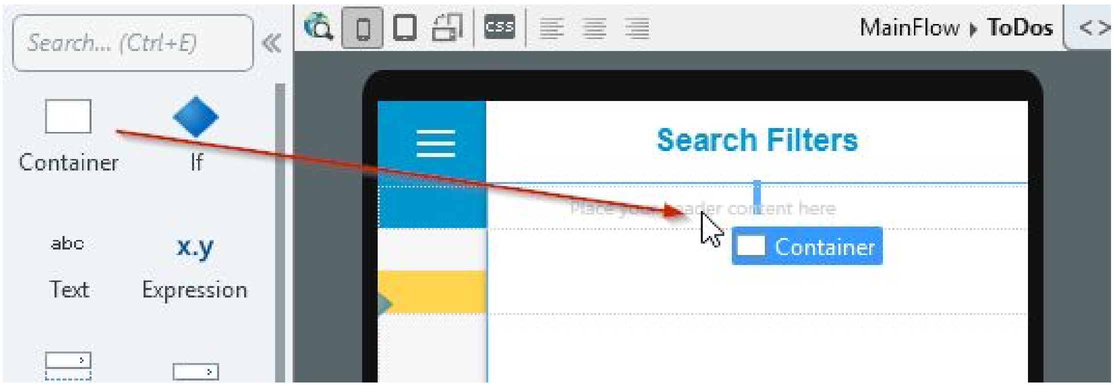
In this step, we are calling the **ToggleSidebar** Action of the Sidebar pattern, that opens / closes the sidebar on the Screen. The **WidgetId** parameter expects the Id of the Sidebar that we are controlling in this step, which is the *SearchSidebar*.



Name	ToggleSidebar
Action	Sidebar\ToggleSidebar
WidgetId	SearchSidebar.Id

2) Define a **Search** input field for the Sidebar that filters the list of ToDos for their **Title** or **Notes**. Make sure that when the Search is made, by clicking on a Button, the **GetToDos** Aggregate is executed again, to update the list of To Dos returned from local storage.

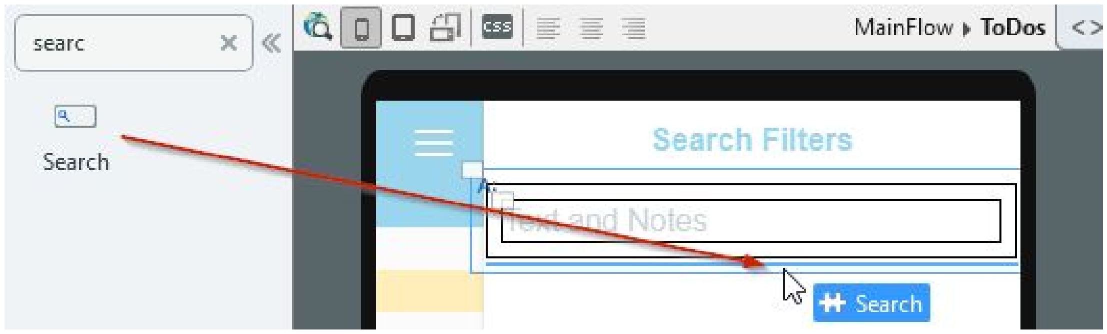
   a) Return to the **ToDos** Screen, drag a new **Container** and drop it inside the **Content** placeholder of the Sidebar.

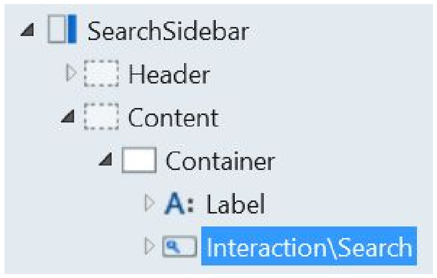


   b) Drag a **Label** widget and drop it inside the Container created in the previous step.

   c) Change the **Text** inside the Label to *Text and Notes*. Align it to the *left*.

   d) Drag a **Search** widget and drop it inside the same Container. Make sure that it is not inside the Label Widget.





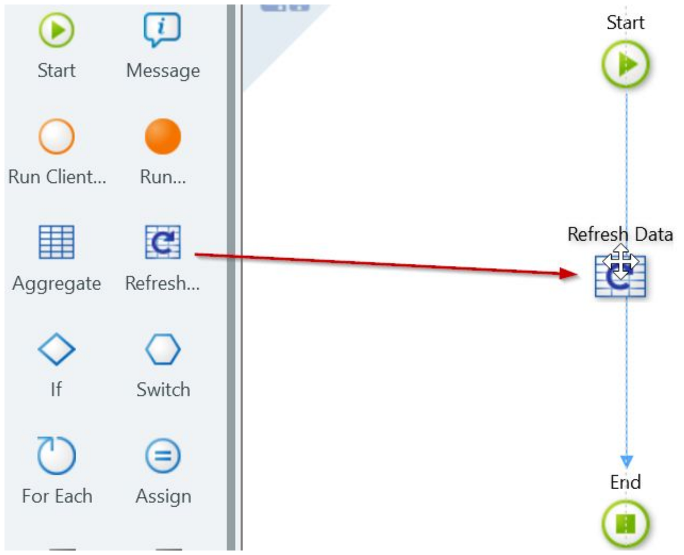
e) Select the **Input** inside the Search widget, expand the drop down of suggestions for the **Variable** property and choose *(New Local Variable)*.

f) Rename the new Local Variable to *SearchKeyword*.

g) Drag a new **Button** and drop it below the Search widget on the Sidebar. Set the **Text** inside it to *Search*.

h) Double-click on the Button to create a new Action. Set its **Name** to *Search*.

i) Drag a **Refresh Data** statement and drop it on the Action flow.



j) In the **Select Data Source** dialog, choose the **GetToDos** Aggregate.

NOTE: The **Refresh Data** statement re-executes the Aggregate. At this point, our Aggregate does not filter the results based on the search keyword. This will be done in the following steps.

k) Drag a **Run Client Action** statement and drop it between the Refresh Data and the End of the flow.

l) Select the **ToggleSearchSidebar** Action of the ToDos Screen.

NOTE: The **Search** Client Action is at this point calling another Client Action inside the same Screen. This kind of behavior enables developers to create smaller and reusable flows that can depend on each other.

3) Modify the **GetToDos** Aggregate to use the **SearchKeyword** Local Variable on its Filters. This way, when the Aggregate is executed again after the Search, it will use the value introduced by the user in the input to filter its result.

   a) Open the **GetToDos** Aggregate in the ToDos Screen.

   b) In the Filters tab, click the **+Add Filter** and write the following expression

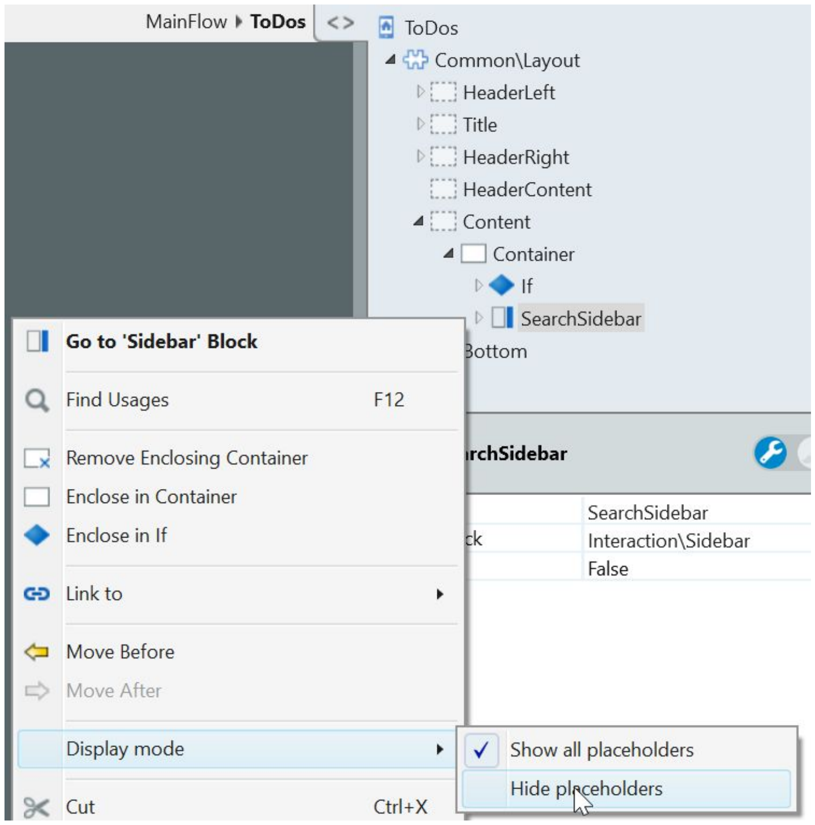
      LocalToDo.Title like "%" + SearchKeyword + "%" or LocalToDo.Notes like "%" + SearchKeyword + "%"

      This expression will define a Filter on the Aggregate to search for LocalToDos with the SearchKeyword included on the Title and Notes. Meaning that if a user searches for *milk*, any ToDo with the keyword *milk* on the Title (or Notes) will be returned (for instance, *Buy milk*).

4) On the **ToDos** Screen, replace the **plus** icon on the top right corner for a *search* icon. Then, instead of linking the icon to the **ToDoDetail** Screen, make sure that the Sidebar opens, using the **ToggleSidebar** Action.

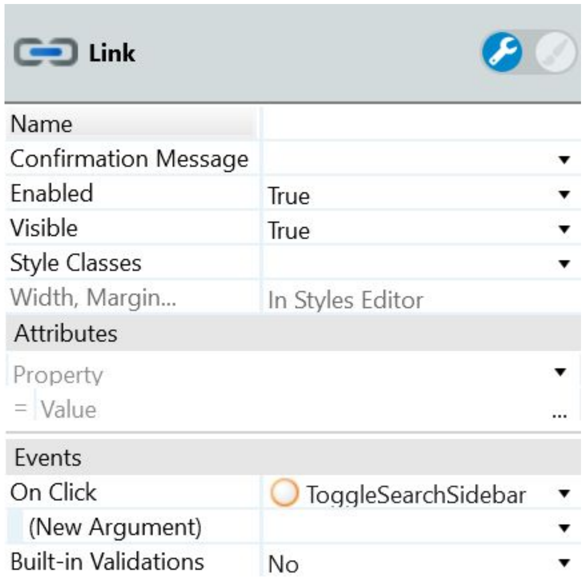
   a) On the **ToDos** Screen, open the Widget Tree by clicking the **<>** button. Then, right-click on the **Sidebar** and choose *Display mode > Hide placeholders*.

      This will hide the Sidebar, making the content of the ToDos Screen visible. This can be extremely helpful for continuing to edit the other parts of the Screen where the Sidebar is.



b) Double click the **plus** Icon on the top right of the Screen and choose the *search* icon.

c) Select the **Link** surrounding the Icon and change the **OnClick** property to the *ToggleSearchSidebar* Action. Delete the unused parameter after the change.



Link

Name	
Confirmation Message	
Enabled	True
Visible	True
Style Classes	
Width, Margin...	In Styles Editor
Attributes	
Property	
= Value	...
Events	
On Click	ToggleSearchSidebar
(New Argument)	
Built-in Validations	No

5) Publish the module and test the Sidebar on the ToDos Screen.

   a) Click the 1-Click Publish Button to publish the module to the server.

   b) In the ToDos Screen, swipe the Sidebar open from the right edge of the Screen.

   c) Type any search keyword that would filter the ToDos returned.

   d) Make sure the list of ToDos is updated and that the Sidebar closes.

   e) Open it again by using the icon on the ToDos Screen.

   f) Close the Sidebar by clicking on the Button. Make sure it closes.

Add a Priority filter to the Sidebar

In this section, the Sidebar filters will be expanded to also allow to search based on the To Do Priority. This new filter will allow users to see only To Dos with a specific Priority or all of them.

1) Add a **ButtonGroup** to the Sidebar to filter the ToDos by priority. Make sure the ButtonGroup appears between the Input for Title and Notes and the Search Button. The ButtonGroup should have four options: *All*, *Low*, *Medium* and *High*.

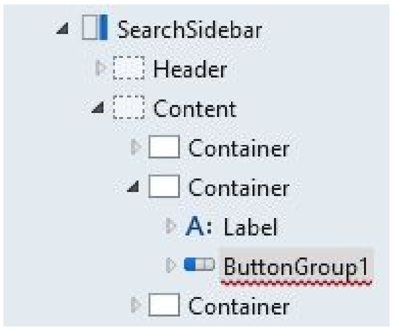
    a) In the **ToDos** Screen, open the Widget Tree, then right-click the Sidebar and choose the *Display mode > Show all placeholders*

    b) Drag a **Container** widget and drop it between the Search and the Container with the Search button.

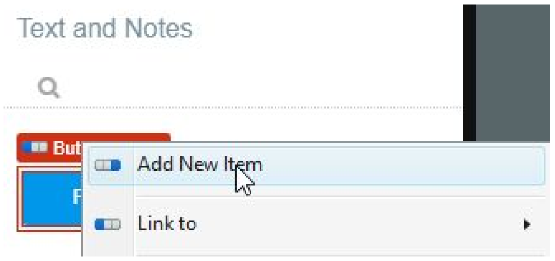
    c) Drag a **Label** widget and drop it inside the Container and align it to the left.

    d) Set the **Text** of the Label to *Priority*.

    e) Drag a **ButtonGroup** and drop it inside the Container and below the Label.

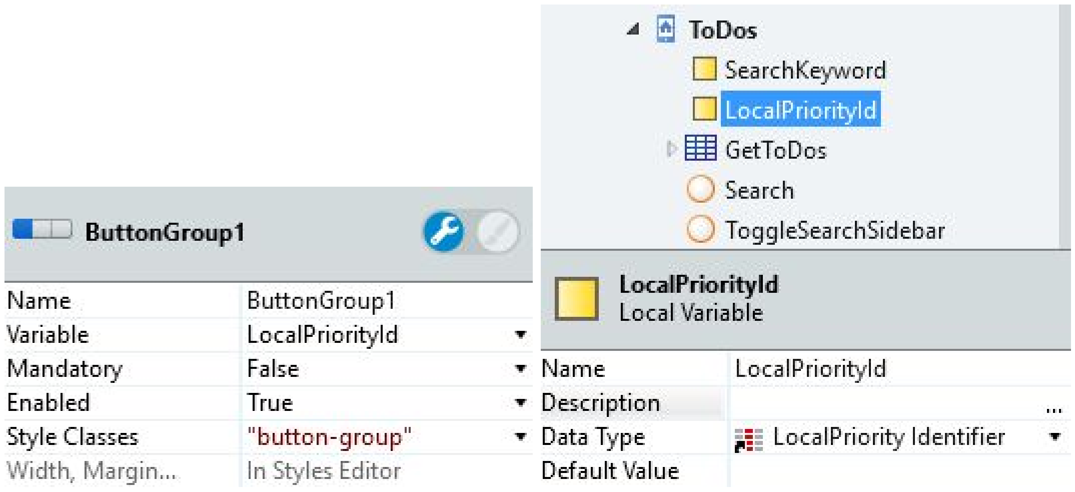


    f) Right-click the Button Group and choose *Add New Item*.



    g) With the ButtonGroup selected, set the **Variable** property to *(New Local Variable)*.

    h) Change the variable **Name** to *LocalPriorityId* and make sure its **Data Type** is set to *LocalPriority Identifier*.

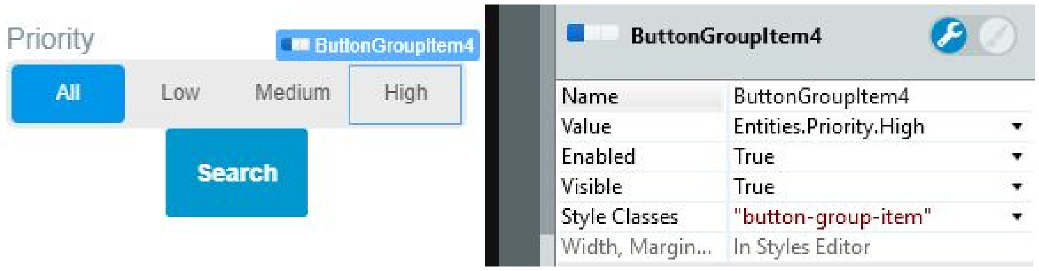


Name	ButtonGroup1
Variable	LocalPriorityId
Mandatory	False
Enabled	True
Style Classes	"button-group"
Width, Margin...	In Styles Editor

Name	LocalPriorityId
Description	
Data Type	LocalPriority Identifier
Default Value	

i) Change the **Text** of the first button of the group to *All* and set its **Value** to *NullIdentifier()*

j) Set the remaining buttons **Text** to *Low*, *Medium* and *High*, with the respective **Values** of *Entities.Priority.Low*, *Entities.Priority.Medium* and *Entities.Priority.High*



Name	ButtonGroupItem4
Value	Entities.Priority.High
Enabled	True
Visible	True
Style Classes	"button-group-item"
Width, Margin...	In Styles Editor

2) Modify the **GetToDos** Aggregate to filter the results based on the selected priority.

   a) Open the GetToDos Aggregate and open the Filters area.

   b) Create another Filter and set its Expression to

      LocalPriorityId = NullIdentifier() or LocalToDo.PriorityId = LocalPriorityId

      The first part of the condition checks if the variable that holds the choice made by the user, in the Sidebar filter, is *NullIdentifier()*. This covers the situation where no priority, or the **All** priority, is chosen. Otherwise, if the condition only considered the second part, no ToDo would be returned, when the **LocalPriorityId** variable had *NullIdentifier()*, since all ToDos must have a Priority.

   c) Click **Done** to close the Expression Editor.

3) Publish the module and test the Sidebar on the ToDos Screen.

   a) Click the **1-Click Publish** Button to publish the module to the server.

b) Confirm that all ToDos appear at this point.

c) In the ToDos Screen, open the **Sidebar**.

d) Switch the priority filter to **Low** and click on the **Search** button.

e) Make sure the list of ToDos only displays the ToDos with Low priority.

Add a Category filter to the Sidebar

In this section, the Sidebar filters will be expanded to also allow to search based on the Categories. For each Category, a checkbox will be displayed and enable users to filter the To Dos based on a set of Categories.

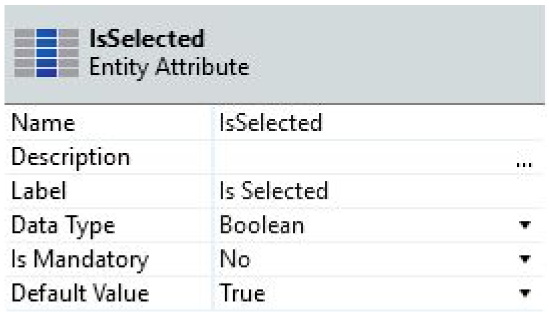
We will start by adding a new attribute to the **LocalCategory** Entity, called *IsSelected*. This attribute will be useful for keeping track of which categories are selected in the search filter. When we do the data synchronization, the local categories are all deleted, before being created again with fresh data that comes from the server. This will clear the search filter in every sync. This way, we can use this attribute to tweak the synchronization, to avoid that to happen.

1) Add a new *IsSelected* attribute to the **LocalCategory** Entity.

    a) Switch to the **Data** tab and locate the **LocalCategory** Local Storage Entity.

    b) Add a new attribute and name it *IsSelected*. Make sure the **Data Type** is set to *Boolean*.

    c) Set the **Default Value** to *True*.



IsSelected
Entity Attribute

Name	IsSelected
Description	...
Label	Is Selected
Data Type	Boolean
Is Mandatory	No
Default Value	True

NOTE: The *IsSelected* attribute will act as a device setting and be persisted in the Local Storage Entity. Later on, we will change the synchronization of Categories to take into account this new attribute that only exists in Local Storage.

2) Add a new filter to the Sidebar to allow filtering by Categories. The Sidebar should display a checkbox for each existing category. Whenever a checkbox of a category is ticked / unticked, the *IsSelected* attribute of the **LocalCategory** attribute is updated.

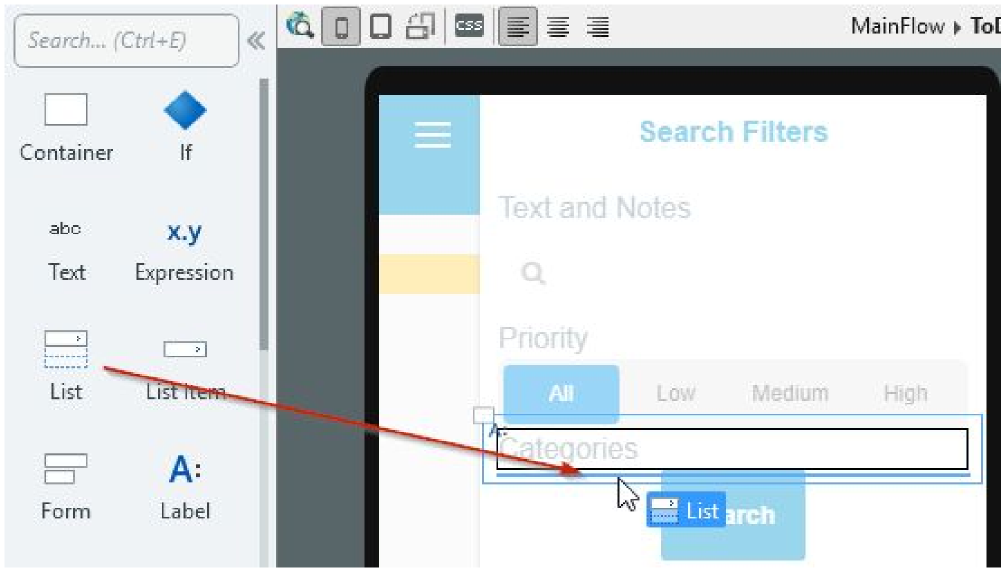
    a) Open the **ToDos** Screen from the Interface tab.

    b) Drag a **Container** widget and drop it between the Priority and the Container with the Search button.

    c) Drag a **Label** widget and drop it inside the Container and align it to the left.

d) Set the **Text** of the Label to *Categories*.

e) Drag a **List** widget and drop it below the Label, but inside the Container.



f) Right-click the **ToDos** Screen and select *Fetch Data from Local Storage*.

g) Drag the **LocalCategory** Entity and drop it inside the Aggregate.

h) Return to the **ToDos** Screen, and set the **List** widget **Source** to the *GetLocalCategories* Aggregate.

i) Drag a **Container** widget and drop it inside the List widget, then align it to *left*.

j) Drag a **Checkbox** widget and drop it inside the Container created above, and set the Variable property to

GetLocalCategories.List.Current.LocalCategory.IsSelected

This will set the Checkbox to be selected or not selected, depending on the value of the new attribute.

k) Drag an **Expression** and drop it next to the Checkbox. Set the Expression to

GetLocalCategories.List.Current.LocalCategory.Name

l) Select the Checkbox and in the **On Change** property select *(New Client Action)*. Set its name to *CheckboxOnChange*.

m) Inside the flow of the new Client Action, drag a **Run Client Action** and select *UpdateLocalCategory* Entity Action.

n) Set the **Source** property to

GetLocalCategories.List.Current.LocalCategory

NOTE: The **CheckboxOnChange** Client Action will be executed whenever the user checks or unchecks a Category checkbox. The state of the checkbox is immediately saved in the **IsSelected** attribute of the **LocalCategory** Entity.

---

3) Change the **GetToDos** Aggregate to only return categories whose **IsSelected** attribute is set to *True*.

   a) Open the **GetToDos** Aggregate.

   b) In the Sources tab, click the **+Add Source** and in the dialog select LocalCategory.

NOTE: The new source is added along with a Join. In this case a **Only With** join because the **CategoryId** attribute is mandatory in the **LocalToDo** Entity.

---

   c) In the Filters tab, add a new filter with the following Expression:

      LocalCategory.IsSelected

4) Publish the module and test the Categories filter on the Sidebar of the ToDos Screen.

   a) Click the **1-Click Publish** Button to publish the module to the server.

   b) Confirm that all ToDos appear at this point.

   c) In the ToDos Screen, open the **Sidebar**.

   d) Uncheck some of the **Categories** and click on the **Search** button.

   e) Make sure the list of ToDos only displays the ToDos with the checked categories.

Change Synchronization to keep Category filters

The previously created Category filters will now be improved to act as a device setting. This setting, that is currently persisted in the Local Storage, is being lost whenever the synchronization is executed. In this exercise, the synchronization will be changed in such way that the setting is not lost when the synchronization runs.

1) Verify that the Category filters are not kept after the synchronization completes.

   a) Open the application and uncheck some of the checkboxes in the Sidebar, then click **Search** and verify that some To Dos were filtered.

   b) In the **DataManagement** Screen, click the **Sync to Local Storage** button and wait for it to finish.

   c) Navigate back to the ToDos Screen and verify that all checkboxes are ticked in the Sidebar.

   NOTE: The goal of the **IsSelected** is to act as a device setting, therefore the sync should not override the value when the sync occurs. In the following steps the synchronization will be changed to prevent losing the setting.

2) Modify the Categories synchronization logic to ensure that the **IsSelected** attribute state is kept. Instead of a Delete All and Create All, each record obtained from the server needs to be created individually, while keeping the value of the **IsSelected** attribute.

   a) Open the **SyncLocalCategories** Client Action from the Logic tab.

   b) Delete the **DeleteAllLocalCategories** and **CreateOrUpdateAllLocalCategories** Entity Actions.

   c) Drag a **For Each** statement and drop it between the SyncCategories and End. This will help us iterate through the list of categories fetched from the server.

   d) In the **Record List** property choose

      SyncCategories.LocalCategories

   e) Drag an **Aggregate** and drop to the right of the For Each, then create the connector from the latter to the former.

   f) Double-click the Aggregate to open it. Add the **LocalCategory** Entity as a source by dragging it from the Data tab.

   g) In the Filters tab, add a new filter

      LocalCategory.Id = SyncCategories.LocalCategories.Current.Id

Instead of deleting everything from the local storage, here we fetch the Local Category that matches the Id of the Category currently being iterated in the loop.

h) Return to the **SyncLocalCategories** Client Action, then drag an **Assign** and drop to the right of the **GetLocalCategoryById** Aggregate.

i) Create the connector from the Aggregate to the Assign.

j) In the Assign, create a new assignment as follows

SyncCategories.LocalCategories.Current.IsSelected

= *GetLocalCategoryById.List.Current.LocalCategory.IsSelected*

This will make sure that the Categories fetched from the server, will maintain the **IsSelected** property in the Local Storage, which was fetched in the previous Aggregate. Doing this in a loop, guarantees that it is performed for every Category fetched from the server.

k) Drag a **Run Client Action**, and drop it below the Assign. In the dialog choose the *CreateOrUpdateLocalCategory* Entity Action.

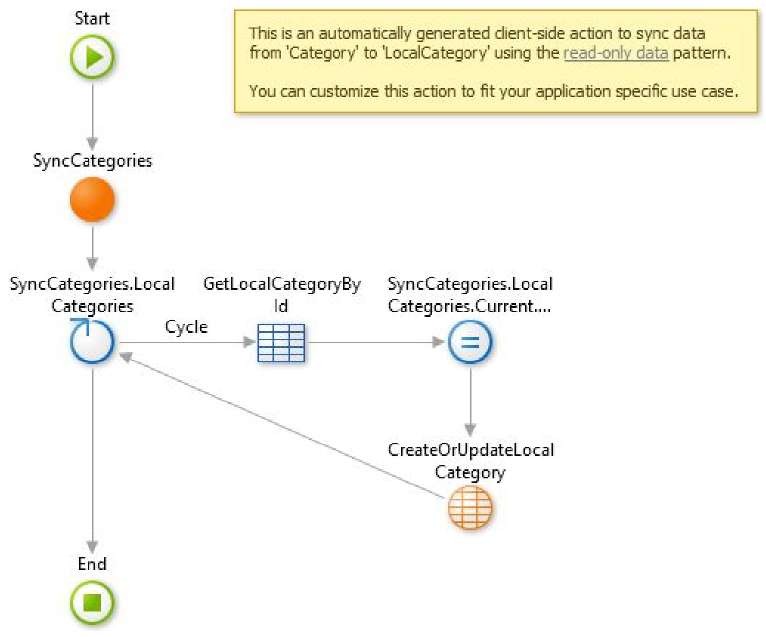
l) Create a connector from the Assign to the Entity Action, and another from the Entity Action to the For Each.

m) Set the Source property of the Entity Action to

SyncCategories.LocalCategories.Current

This will save in the local storage the Local Storage coming from the server, while keeping the IsSelected attribute value.

n) The flow should look like this


Start
SyncCategories
SyncCategories.Local Categories
Cycle
GetLocalCategoryBy Id
SyncCategories.Local Categories.Current....
CreateOrUpdateLocal Category
End

This is an automatically generated client-side action to sync data from 'Category' to 'LocalCategory' using the read-only data pattern.

You can customize this action to fit your application specific use case.

3) Publish the module and test the Categories filter on the Sidebar of the ToDos Screen.

   a) Click the **1-Click Publish** Button to publish the module to the server.

   b) Confirm that all ToDos appear at this point.

   c) In the ToDos Screen, open the **Sidebar**.

   d) Uncheck some of the **Categories** and click on the **Search** button.

   e) Make sure the list of ToDos only displays the ToDos with the checked categories.

   f) In the Data screen, click the *Sync to Local Storage* button and wait for it to finish.

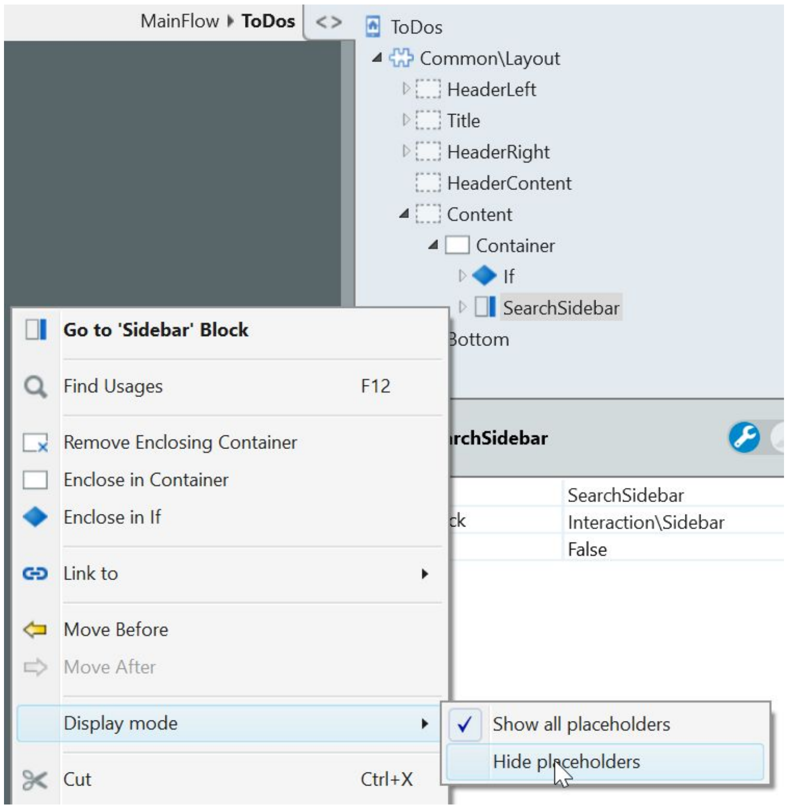
   g) Navigate back to the ToDos screen and verify that all checkboxes state is kept even after sync.

Clear Sidebar Filters

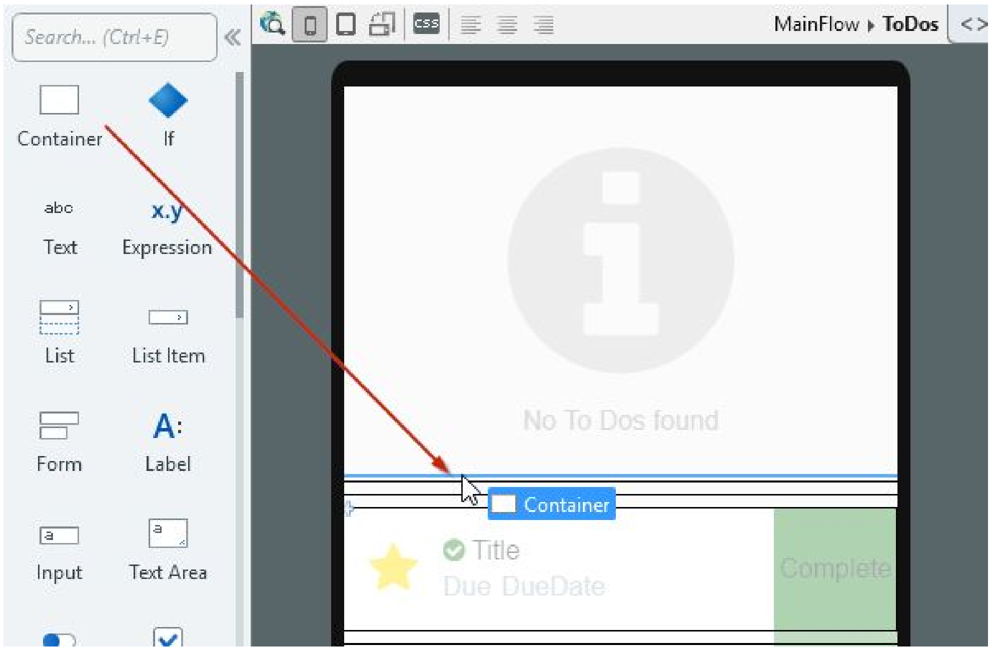
In this exercise, we will add a visual hint that search filters are defined. This visual hint will improve the usability of our app by warning users that filters are active and enable them to clear all filters with one click.

1) Add a **Clear Filters** button to the **ToDos** Screen. The Button should appear right before the list of ToDos and it should be red, with white text and it should trigger a new Client Action.

    a) On the **ToDos** Screen, open the Widget Tree by clicking the **<>** button. Then, right-click on the **Sidebar** and choose *Display mode* > *Hide placeholders*.



    b) Drag a **Container** and drop it just above the List that displays the To Dos.

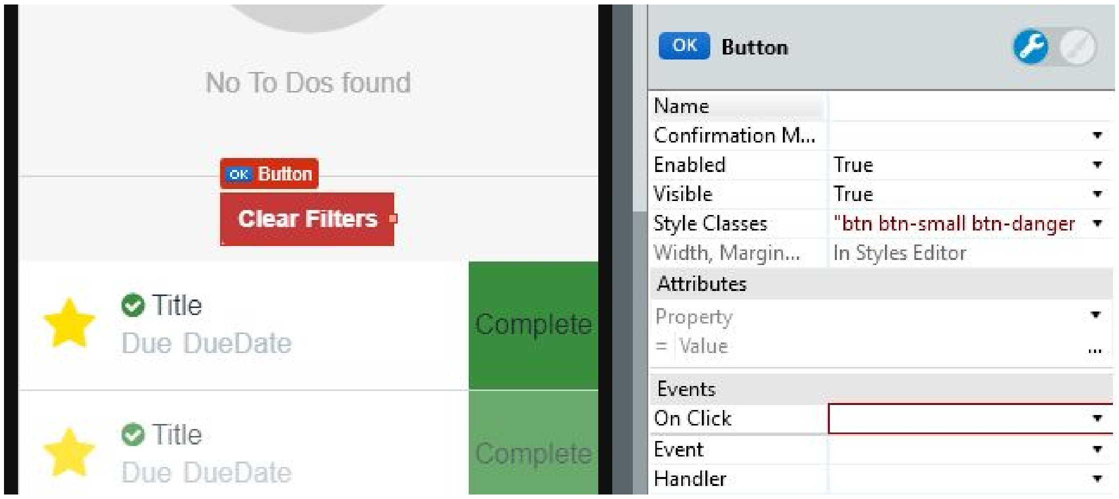


c) Change the **Style Classes** property of the container to *"padding"*.

d) Drag a **Button** and drop it inside the Container created above. Set the Button text to *Clear Filters*.

e) Select the Button and set the **Style Classes** property to

"btn btn-small btn-danger"



Name	
Confirmation M...	
Enabled	True
Visible	True
Style Classes	"btn btn-small btn-danger
Width, Margin...	In Styles Editor
Attributes	
Property	
= Value	...
Events	
On Click	
Event	
Handler	

f) Double-click the Button or select *(New Client Action)* in the **On Click** Event property to create a new Screen Action.

2) Create the logic that will enable the **Clear Filters** Button to clear the Text and Notes, and Priority filters.

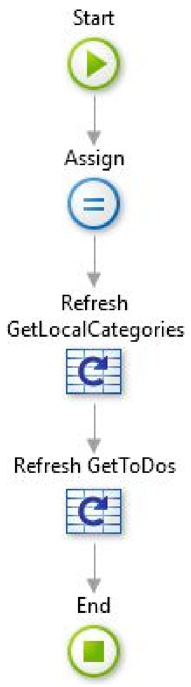
a) Drag an **Assign** statement and drop it in the new Action flow. Then, define the following assignments

SearchKeyword = ""

LocalPriorityId = NullIdentifier()

b) Drag a **Refresh Data** and drop it after the Assign. Set the **Data Source** to *GetLocalCategories*.

c) Drag another **Refresh Data** and drop it after the previous. Set its **Data Source** to *GetToDos*.



3) Publish the module and test the Clear Filters button.

a) Click the **1-Click Publish** Button to publish the module to the server.

b) Confirm that all ToDos appear at this point.

c) Open the **Sidebar**, type some text (e.g. Buy milk) or select on of the priorities then click Search.

d) At this point you should see only some of the To Dos.

e) Click the **Clear Filters** button and notice that all To Dos appear.

4) Create a new Aggregate on the **ToDos** Screen, to verify what categories are not selected, and update the logic of the **Clear Filters** button to select all categories, when the button is clicked.

a) Right-click the ToDos Screen and select **Fetch Data from Local Storage** to create a new Aggregate. Set the Aggregate name to *GetLocalCategoriesNotSelected*.

b) From the Data tab, drag the **LocalCategory** Entity to the Aggregate.

c) Add the following Filter to the Aggregate

   LocalCategory.IsSelected = False

d) Open the **ClearFiltersOnClick** Client Action.

e) Drag a **For Each** and drop it between the existing Assign and the first Refresh Data.

f) Set the **Record List** property to *GetLocalCategoriesNotSelected.List*.

g) Drag an **Assign** and drop it to the right of the For Each, then create the Cycle connector from the For Each to the Assign.

h) In the Assign, create the following assignment

   GetLocalCategoriesNotSelected.List.Current.LocalCategory.IsSelected

   = True

i) Drag the **UpdateLocalCategory** Entity Action from the Data tab, then create a connector from the Assign to this Action.

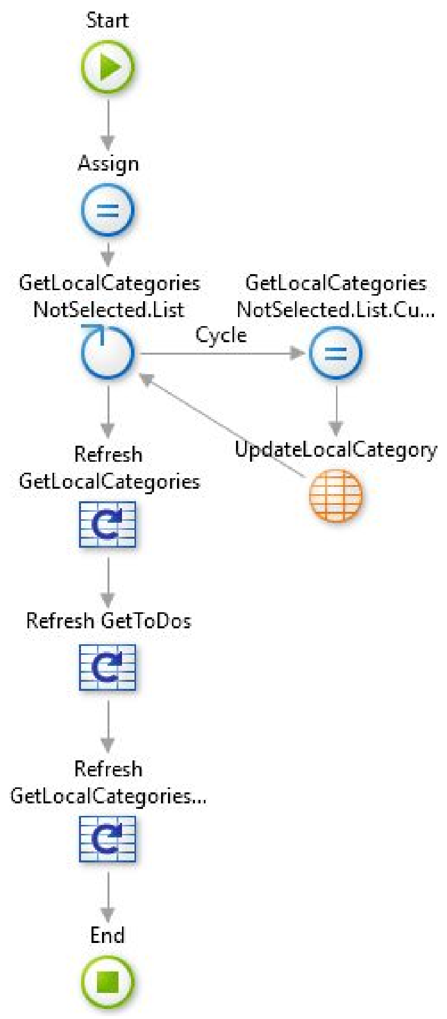
j) Set the **Source** property of the **UpdateLocalCategory** Entity Action to

   GetLocalCategoriesNotSelected.List.Current

   This will make every Category that was returned from the Aggregate to be set as selected.

k) Close the For Each by creating the connector from the UpdateLocalCategory back to the For Each.

l) Drag a **Refresh Data** and drop just before the End. Set the **Data Source** to *GetLocalCategoriesNotSelected*. This is important to run the Aggregate again, so that all the Categories may show up.


Start
Assign
GetLocalCategories NotSelected.List
GetLocalCategories NotSelected.List.Cu...
Cycle
UpdateLocalCategory
Refresh GetLocalCategories
Refresh GetToDos
Refresh GetLocalCategories...
End

5) Make the **Clear Filters** button visible only when at least of filter is active.

a) Return to the **ToDos** Screen, and select the Container surrounding the Clear Filters button.

b) Change the **Visible** property to

SearchKeyword <> "" or LocalPriorityId <> NullIdentifier() or GetLocalCategoriesNotSelected.List.Length > 0

This condition verifies, in the search filter, that the search for text and notes input field is empty, no Priority is selected and no Categories were returned from the **GetLocalCategoriesNotSelected** Aggregate, meaning that all Categories are selected.

c) Open the **CheckboxOnChange** Action, then drag a **Refresh Data** and drop it after the existing **UpdateLocalCategory** Entity Action.

d) Set the **Data Source** to *GetLocalCategoriesNotSelected*.

6) Publish the module and test that the Clear Filters button only appears when filters are active.

    a) Click the **1-Click Publish** Button to publish the module to the server.

    b) If the Clear Filters is displayed, click it.

    c) All To Dos should be visible at this point.

    d) Open the **Sidebar**, try each of the filters (Text and Notes, Priority, Categories) individually click the Search button.

    e) The Clear Filters button show be visible whenever any of the filters is set.

End of Lab

In this exercise lab, we created a Sidebar to help us filter the list of To Dos displayed on the ToDos Screen.

The sidebar allows searching for text, notes, priority and categories. This will influence the ToDos that appear on the list, which required refreshing Aggregates when the search is made, to update the results.

Then, we tweaked the synchronization of categories, to avoid that the selected categories would be cleared after each sync. For that, we created a new attribute on the LocalCategory Entity, **IsSelected**, to save if the Category is selected. Then, in the synchronization logic, we stopped deleting all the Local Categories. In its place, we created some logic to make sure that the **IsSelected** value would not be lost.

Then, we added a new Button to clear the priority and the text and notes filter.