Lifecycle exercise

Table of Contents

knowledge@outsystems.com

www.outsystems.com

Outline

In this exercise, we will focus on:

- How to use the Screen lifecycle events to add behaviour to a screen
- Manipulating data when retrieved
- Using the On Parameters Changed event to communicate update data a Block displays

Upon completion, we'll have fixed an application so a block can react to changes to the screen, and only display certain UI elements after data has been retrieved, as well as limiting the ability of a user to show the context menu on a specific Screen.

Resources

The Events Frontend application needs to be installed, as you will be modifying it.

Scenario

The application provided allows users to register alarming situations happening in different countries, and classify them by severity (or whether they have been addressed). Unfortunately, the chart on the dashboard is not being updated when there are changes and it would improve readability if we could see the total number of alarms at a glance

Also, the app has a sample screen that should be limiting the user capacity to bring up the context menu but is not.

These are the situations you will need to address.

How-To

In this section, we'll provide step-by-step instructions for you to fix the application provided.

Getting Started

To start this exercise, we first need to make sure the **EventsFrontend** is installed. You will be working on its single module, **EventsDemo**.

Although the **Alarms** screen is functional, there is room for improvement. Let's address each concern one at a time.

as the **AlarmChart** Block is not being refreshed whenever we add a new Alarm situation.

Our Block and Screen is ready, now let's add some extra event handling to our application to improve the overall user experience.

Aggregate events

Although the **AlarmChart** block is providing a visual breakdown of alarms, there is no easy way of seeing a grand total. That will be our first task, add a new counter to display the total number of alarms.

Let's start by calculating the total number of alarms. In order to do that we will use the information from the **GetAlarmCountByCountryAndLevel** aggregate, but we can only do it after data is fetched.

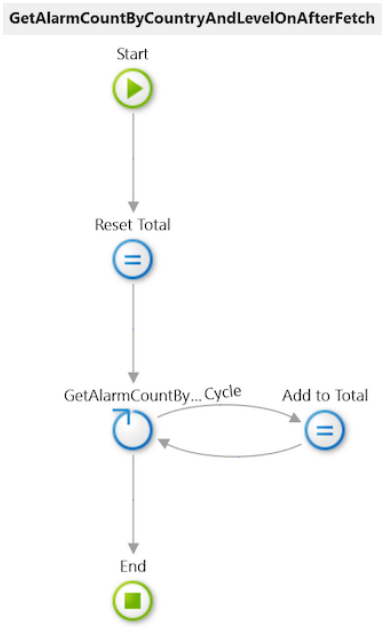
1) Add a new local variable to the **AlarmChart** Block, name it **Total** and set its type to Long Integer.

2) Select the **GetAlarmCountByCountryAndLevel** aggregate and set its *On After Fetch* event handler to a *New Client Action*

3) Edit the new client action's flow to:

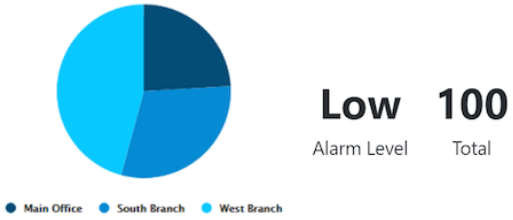
    - Reset the **Total** variable to zero
    - Iterate through the Aggregate's result and add the Count attribute to the **Total** variable

- Your action flow should resemble the following:



4) Create a new Container to the right of the existing Container displaying the alarm level, and mimic it's appearance:

   - Place a new *Expression* widget with its *Value* property set to the **Total** variable and the *Example* property set to `100`
   - Place a *Text* widget with it's *Text* property set to `"Total"`
   - The **AlarmChart** Block should look similar to the following:



5) Publish your changes and check how the new Alarms dashboard looks on the browser.

Extra challenge!

Make sure the PieChart and the Total expression are only visible when the aggregate data is returned.

Tip: use the *IsDataFetched* and *HasFetchError* properties of the aggregate.

Block events

The **AlarmChart** Block is being used on our **Alarms** dashboard screen, and its input parameter is bound to a screen variable that is changed when the user chooses a different alarm level.

Unfortunately, those changes are not being handled by the Block, and we need to address that.

1) Set the *On Parameters Changed* event handler of the **AlarmChart** Block to a *New Client Action*

2) Edit the new client action's flow to:

   - Refresh the **GetAlarmCountByCountryAndLevel** aggregate
   - Refresh the **GetAlarmLevel** Data Fetch action
   - Your action flow should resemble the following:

3) Publish your changes and check that when you select a different Alarm level on the dropdown of the **Alarms** dashboard screen the pie chart and info next to it are updated.

Extra challenge!

Make sure the **AlarmChart** data displayed is also updated whenever you create or edit an Alarm.

Tip: If an input parameter's value changes the Block's *On Parameters Changed* event will be triggered, regardless of whether that input parameter is actually relevant to the block's behaviour.

Screen events

The application also has a bit of an exploratory vein, with an extra screen where we want our users to be unable to right-click and show the context-menu.

Part of the work was already done for us, there are two client actions already implemented that disable and enable the context-menu of the screen. The concern here is that whenever the user arrives at the screen we should disable the context-menu, but this change needs to be undone when the user leaves the screen.

1) Select the **ContextMenu** screen and choose **OnReady** from its *Event* dropdown.

2) Edit the newly created event handler to call the **DisableContextMenu** client action

3) Select the **ContextMenu** screen again and now choose **OnDestroy** from its *Event* dropdown.

4) Edit the newly created event handler to call the **EnableContextMenu** client action

5) Publish your changes and check that when you navigate to the **ContextMenu** screen you cannot right-click anywhere, but when you navigate back to the **Alarms** screen you can

Note: Enabling the context menu when the user leaves the screen is not needed in order to observe the correct behavior, but failing to do so may lead to performance deterioration as resources aren't being released.

5. Conclusion

Everytime any of the **AlarmChart**'s input parameters are updated, the block's *On Parameters Changed* event is triggered.

In particular this happens on our **Dashboard** screen when:

- The dropdown is used to filter data
- (Challenge) A new Alarm is created

Also, in order to avoid costly extra queries to the server, the **Total** is calculated locally after the aggregate's data is fetched.

Finally, our application's **ContextMenu** screen is now blocking the context menu from being displayed, guaranteeing that changes are not impactful once you leave the screen.

Thank you!