# Web Application Process using with Robusta RPA

- In this tutorial, we will show you how to design a sample process that includes Web Application Process with Robusta RPA, step by step.
- In this tutorial we will do a price search from a flight ticket price comparison website that we have determined. For this, we will first ensure that the Booking.com website is opened, and a flight search will be made according to the criteria we have determined. We will find the best match from the incoming search results, assign the price information to a variable, and save the process as successful and end our process. In cases where flight data or the best price match is not found, we will save the process result as failed and end our process.

- We started our process with the Open activity under the Application section to open the Booking.com website. Just drag and drop the activities you want to use into the design area
- In the Name field, we wrote the web site name after the default Open to explain what is done in this step. When we similarly enter a value in the Name field for all the steps, it becomes easier to understand which operations are performed through the process flow.
- Let's enter a reference value without spaces in the Application Name field. This reference will be available to select in the later activities to determine in which application the operation will be performed. Since we will use Chrome as an application, let's choose the Chrome option from the Type field.



| Applications > Open Activity | |
|---|---|
| Name | Open Booking.com |
| *Application Name | bookingDemo |
| *Type | CHROME |
| *Url | https://booking.kayak.com/ |
| Timeout (sec) | 120 |
| Wait until load | ✅ True |
| Maximize | ✅ True |

P.S.1: In this field special character constraints is important, only letters and numbers are recommended
P.S.2: This field will explain later on page 16 or component name > *Wait actions: Check if loaded*

- Since we see that the visible area for the flight departure airport on this site is not the same as the area where we will enter the data in, we continued our process with the Mouse Action activity, as we had to click on this area first in order to make visible to enter data.
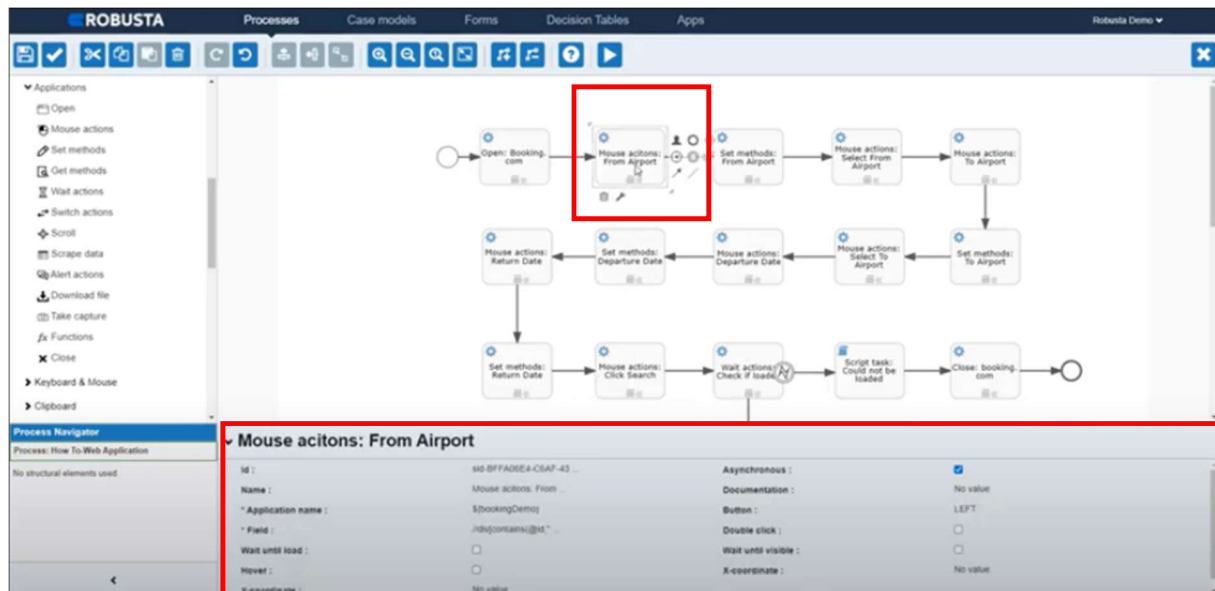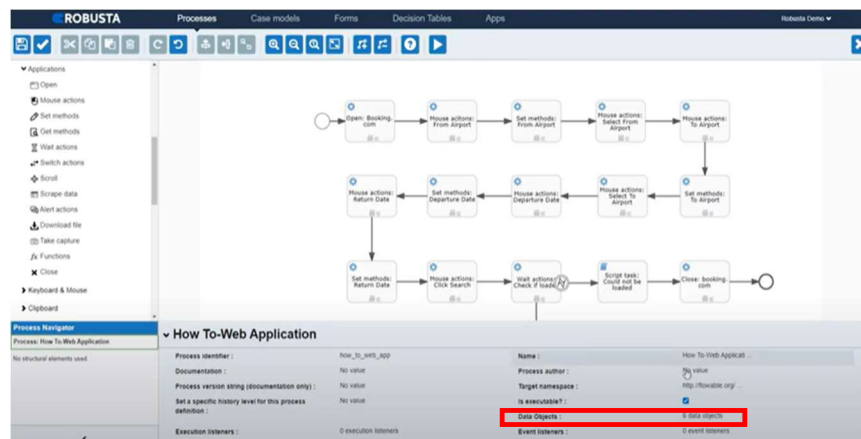


| Applications  > Mouse Actions | |
|---|---|
| Name | Mouse actions : From Airport |
| *Application Name | ${bookingDemo} |
| *Field | //div[contains(@id,"origin-airport-display-inner")] |
| Button | LEFT |

- We have to enter the location information in the html structure of the field that we want to click on the page in the field. For this, we used the Inspector feature of Robusta. In this step, we want to find the location information of the flight departure airport area. For this, we need to click on the "*Field" and then click the New button to add a new page. After choosing Chrome as the type of application we will search in, we entered the link of the "https://booking.kayak.com/" website and clicked the Open button. After waiting for the page to open, we can find the location information of the area we want, by selecting the application from the list and pressing the Inspect button.

To do this, we move our cursor over the relevant field after clicking the button. As you can see the area is marked in yellow. When we wait for 3 seconds on this area, the yellow marking disappears, and the location information is automatically displayed in the RelXPath area. Then, by clicking the OK button, we saved the location information in the field.

Since we have learned how to get Xpath selector data with using Inspector feature of Robusta we will proceed just using correct Xpath without trying to find again in the next components in this tutorial.





- From now we are obtained from the variables on the Data Object screen which we are defined before. We can use these variables by expressing them in the ${} structure that allows us to use them in the process.To reach Data Object window just click white space of canvas and then click "*Data Objects*" area in parameters pane.

- Now we are able to use Data Objects to place them in process easily.
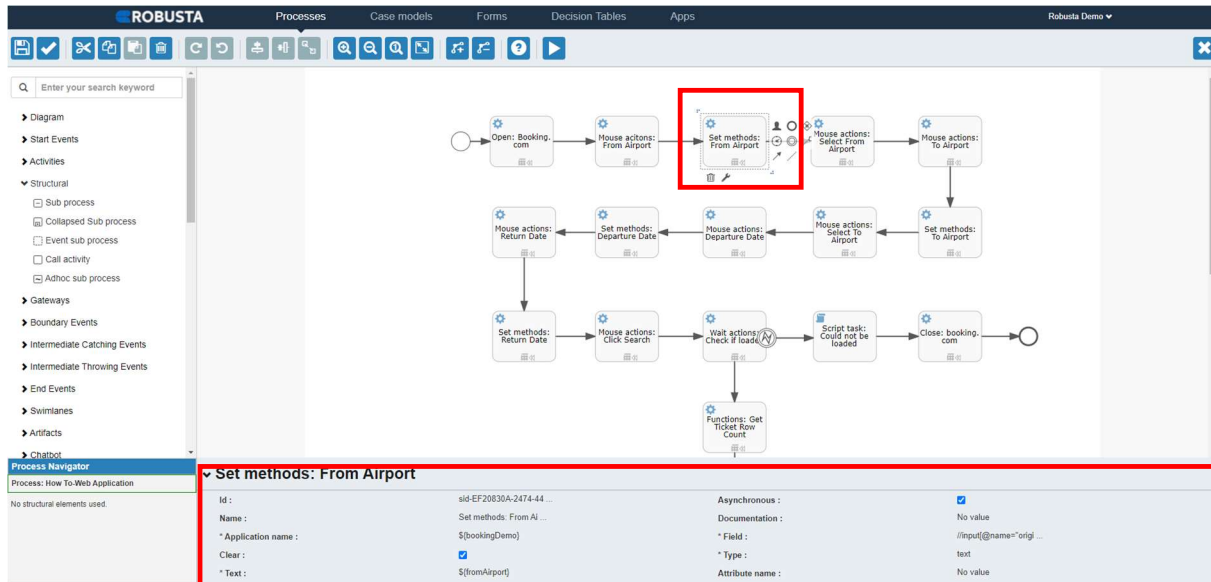


| Applications  > Set Methods | |
|---|---|
| Name | Set methods : From airport |
| *Application Name | ${bookingDemo} |
| *Text | ${fromAirport} |
| *Field | //input[@name="origin"] |
| Wait until visible | ☑ True |
| Clear | ☑ True |
| Button | LEFT |

- With using "*Set Methods*" components we got interact with field that we need to type " *${fromAirport}* " data object to part of web form. We had ensure that the flight departure airport ("*Newyork NY, NYC''*) that we want to search for is written in the relevant field with the Set methods activity.

- ?????In this step, we used another Mouse Action activity, similar to the previous one, to select the airport  in the flight departure point field from the list.?????LOGIC CONTROL
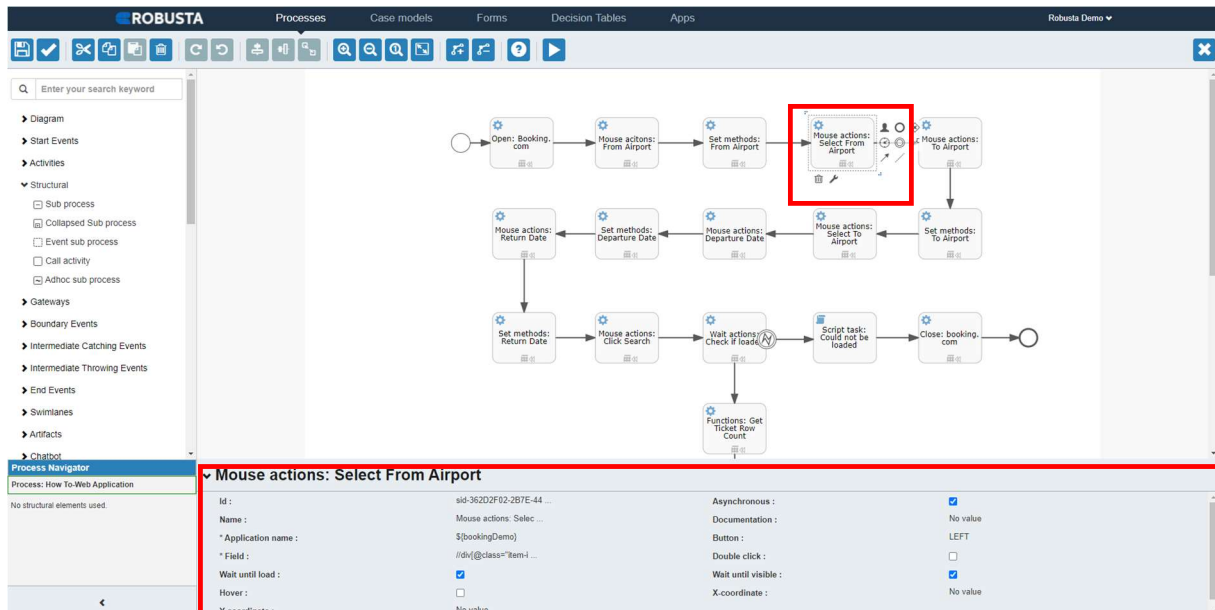


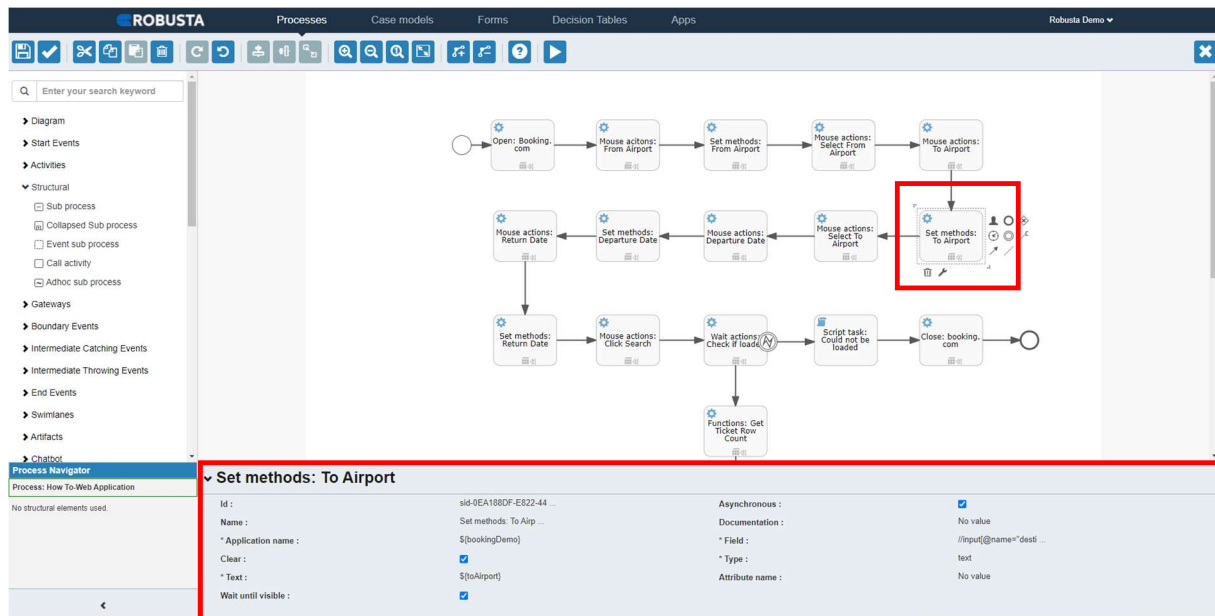| Applications  > Mouse Actions | |
|---|---|
| Name | Mouse actions: Select From Airport |
| *Application Name | ${bookingDemo} |
| *Field | //div[@class="item-info"] |
| Wait until load | ☑    True |
| Wait until visible | ☑    True |
| Button | LEFT |

- As we experience together we continued our process with the Mouse Action activity, as we had to click on this area first in order to make visible to enter data.
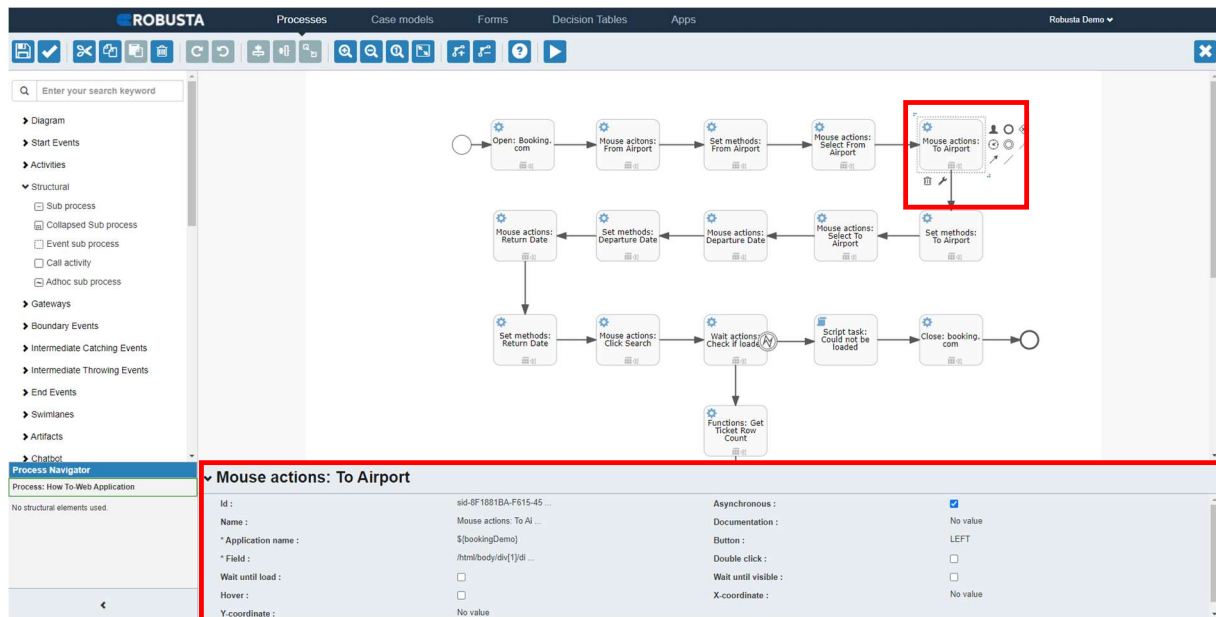


| Applications  > Mouse Actions | |
|---|---|
| Name | Mouse actions : To  Airport |
| *Application Name | ${bookingDemo} |
| *Field | /html/body/div[1]/div[1]/main/div[1]/div[1]/div/div[1]/div/div[2]/section[2]/div/div/div[2]/form[1]/div[1]/div/div[1]/div/div[3]/div/div/div |
| Button | LEFT |

- Now again we have used Data Objects to place variable ( *${toAirport}* ) to field of web form easily.



| Applications > Set Methods | |
|---|---|
| Name | Set methods : To airport |
| *Application Name | ${bookingDemo} |
| *Text | ${toAirport} |
| *Field | //input[@name="destination"] |
| Wait until load | ☑ True |
| Wait until visible | ☑ True |
| Button | LEFT |

- With using "*Set Methods*" components we got interact with field that we need to type " *${toAirport}* " data object to part of web form. We had ensure that the flight departure airport ("*Chicago, IL (CHI)*") that we want to search for is written in the relevant field with the Set methods activity.

- ?????In this step, we used another Mouse Action activity, similar to the previous one, to select the airport  in the flight departure point field from the list.?????LOGIC CONTROL
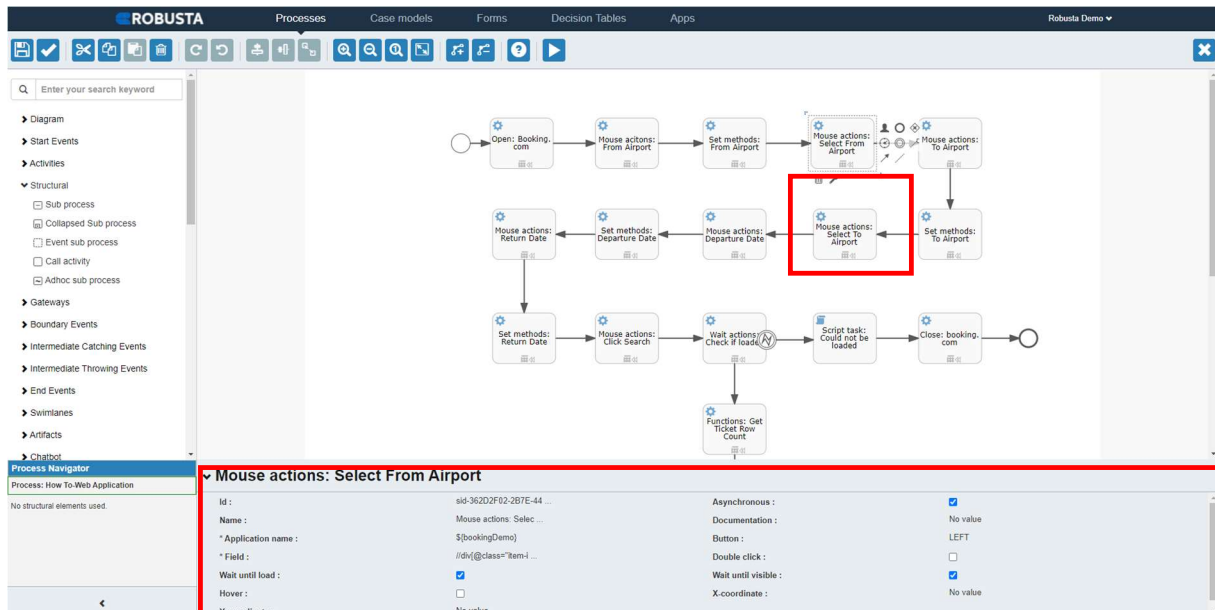


| Applications  > Mouse Actions | |
|---|---|
| Name | Mouse actions: Select To Airport |
| *Application Name | ${bookingDemo} |
| *Field | (//div[@class="item-info"])[2] |
| Wait until load | ✅    True |
| Wait until visible | ✅    True |
| Button | LEFT |

- We continued our process with the Mouse Action activity, as we had to click on this area first in order to enter data.
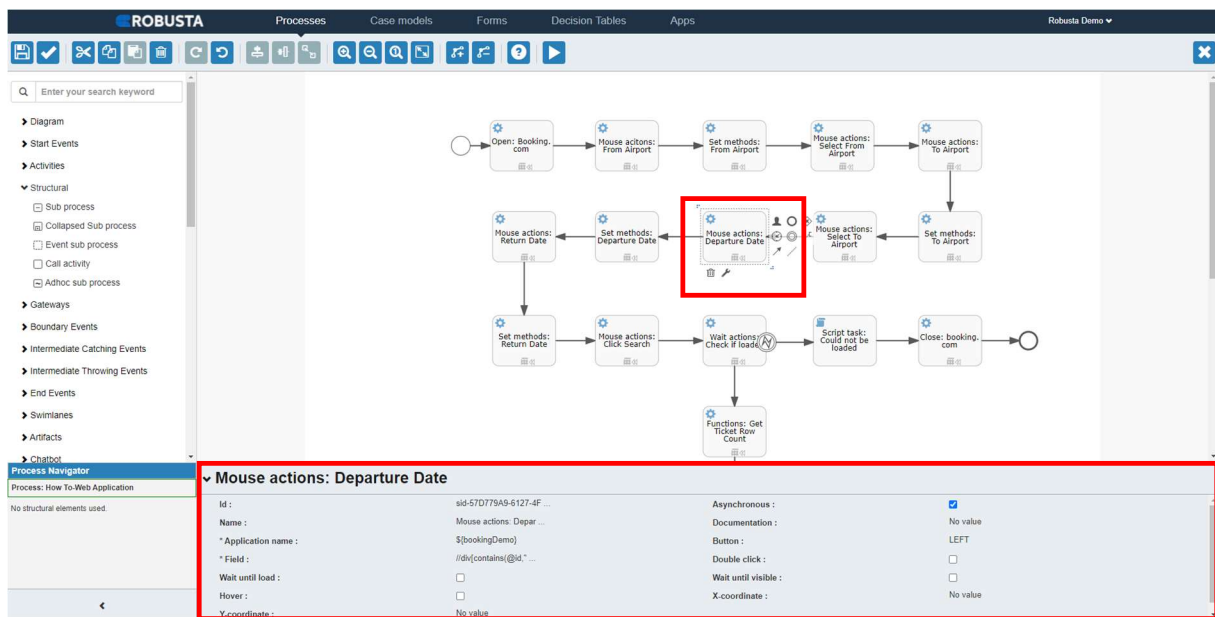


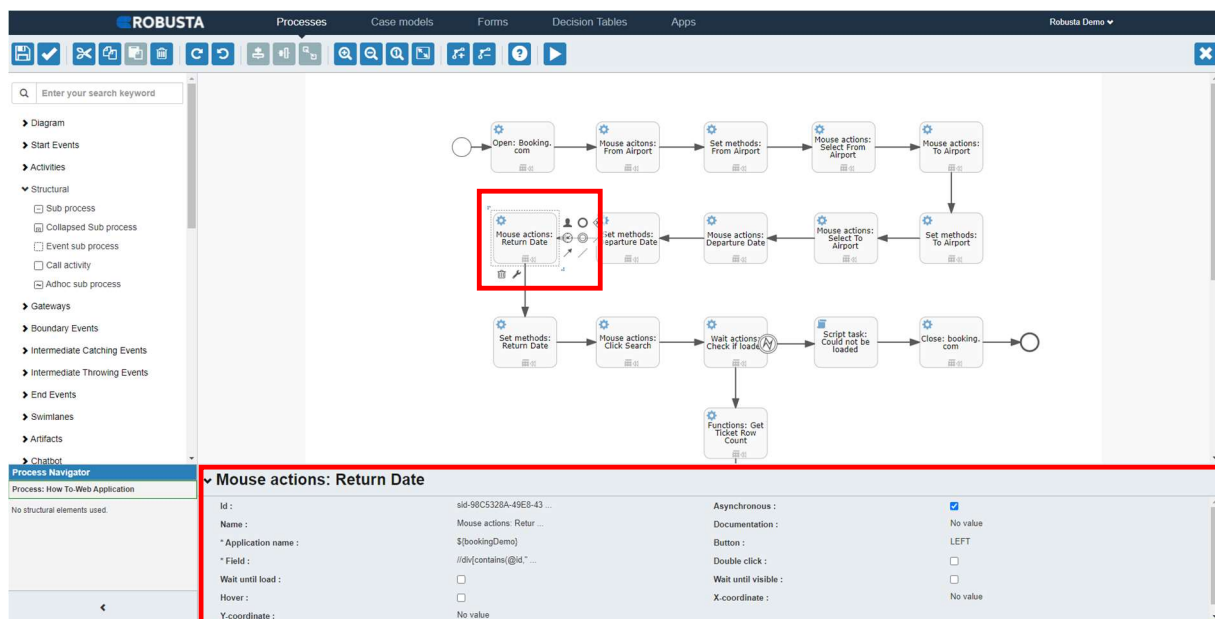| Applications  > Mouse Actions | |
|---|---|
| Name | Mouse actions: Departure Date |
| *Application Name | ${bookingDemo} |
| *Field | //div[contains(@id,"dateRangeInput-display-start-inner")] |
| Button | LEFT |

- Now we have used Data Objects variable ( ${departureDate} ) to place them in process .
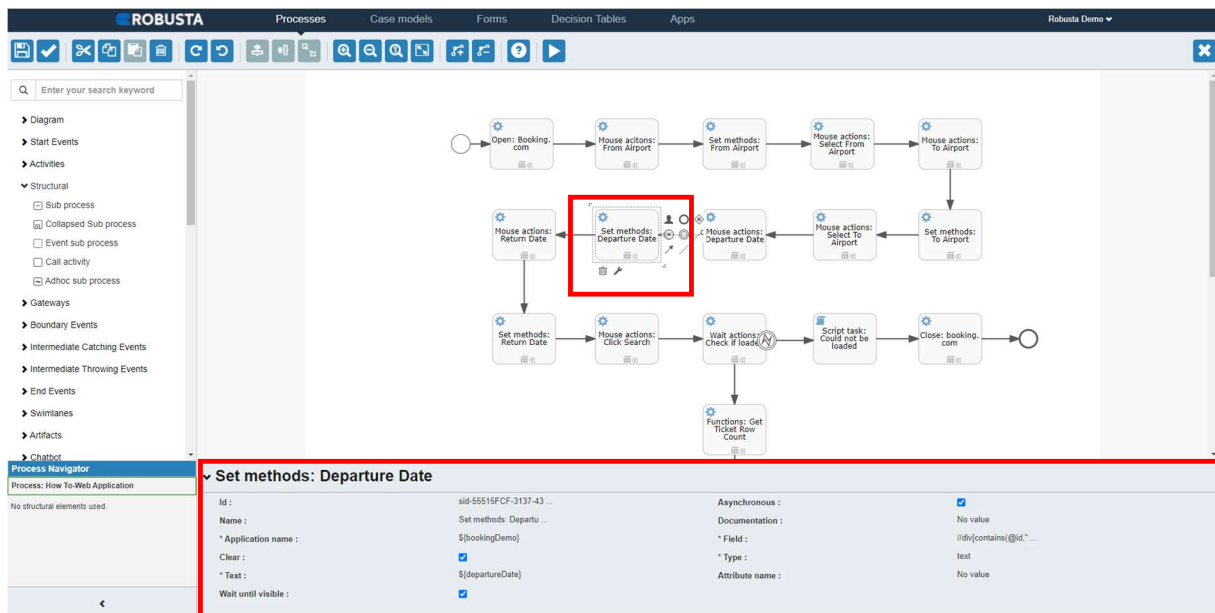


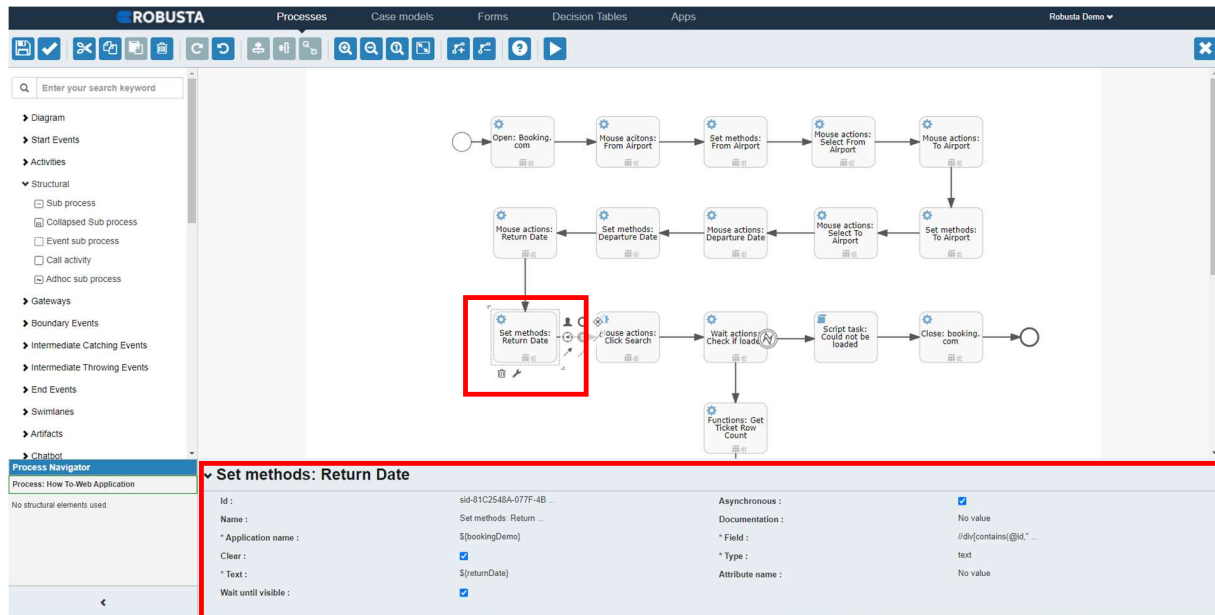| Applications > Set Methods | |
|---|---|
| Name | Set methods: Departure Date |
| *Application Name | ${bookingDemo} |
| *Text | ${departureDate} |
| *Field | //div[contains(@id,"depart-input")] |
| Wait until visible | ☑ True |
| Clear | ☑ True |
| Button | LEFT |

- With using "*Set Methods*" components we got interact with field that we need to type " *${departureDate}* " data object to part of web form. We had ensure that the flight departure date ("*11/09/2021*") that we want to search for is written in the relevant field with the Set methods activity.

- We continued our process with the Mouse Action activity, as we had to click on this area first in order to enter data.



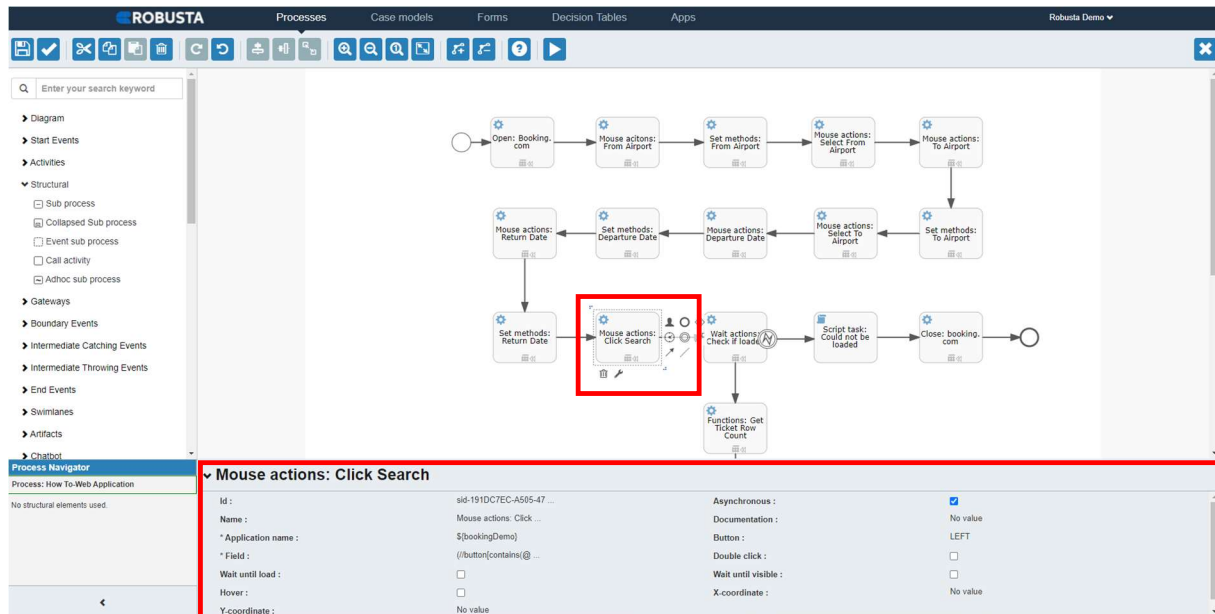| Applications  > Mouse Actions | |
|---|---|
| Name | Mouse actions: Return Date |
| *Application Name | ${bookingDemo} |
| *Field | //div[contains(@id,"dateRangeInput-display-end-inner")] |
| Button | LEFT |

- Now we have used Data Objects variable ( ${returnDate} ) to place them in process .



| Applications  > Set Methods | |
|---|---|
| Name | Set methods: Return Date |
| *Application Name | ${bookingDemo} |
| *Text | ${returnDate} |
| *Field | //div[contains(@id,"return-input")] |
| Wait until visible | ☑  True |
| Clear | ☑  True |
| Button | LEFT |

- With using "Set Methods" components we got interact with field that we need to type  " ${retunDate} " data object to part of web form. We had ensure that the flight departure date ("11/16/2021") that we want to search for is written in the relevant field with the Set methods activity.

- After entering all the criteria what we want, we made sure that the search button was clicked.



| Applications  > Mouse Actions | |
|---|---|
| Name | Mouse actions: Click Search |
| *Application Name | ${bookingDemo} |
| *Field | (//button[contains(@id,"submit")])[1] |
| Button | LEFT |

- On the page that appears after clicking the Search button, we can monitor the loading process from the Loading field on the top left of the page. As you can see, when the page loads, this field is updated and its contents changes. We used this field to check if the search failed or not. For this, we used the Wait action(s) activity, where we can wait until the area we specified on the application is visible or active .
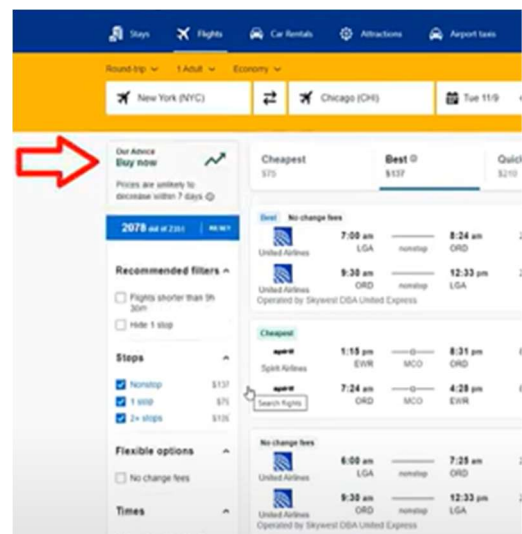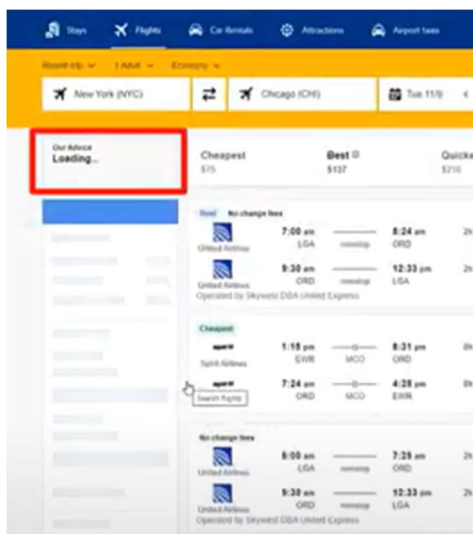
- In this activity, we chose the visible option in the Wait for field, as we wanted to wait until the location field we entered in the field is visible. In this step, we need to return to the Open activity (first component at the first page) for how long we want to wait and enter it in the Timeout field as in seconds. We entered the value 120 in this field, as it can take time to load results on Booking.com. If the results cannot load within the specified time, this step will end with an error.
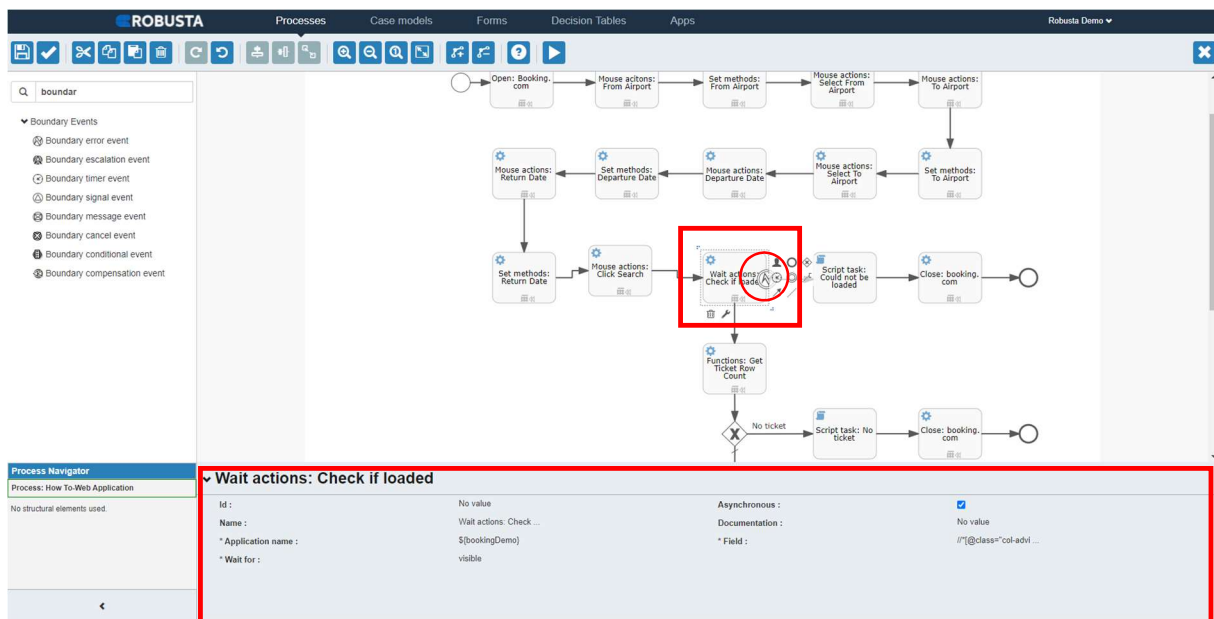


| Applications > Wait Actions | |
|---|---|
| Name | Wait actions: Check if loaded |
| *Application Name | ${bookingDemo} |
| *Field | //*[@class="col-advice"]//*[@aria-busy="false"] |
| *Wait for | visible |



- We want to advance our process from a different flow when this error is received. We can achieve this by adding an error boundary event on the activity. When we drag the error boundary event under the boundary event in the left menu and bring it over the activity, a green frame is displayed around the activity and this means that we can drop the error boundary event and connect it with the activity. Thanks to the error boundary event, the process will continue from here.

- We used the Script Task activity to generate the result message in case the page could not be loaded, and the process continued from the error step. We used the javascript programming language to print the result message and wrote javascript in the Script format field. In this activity, after ensuring that the ticketPrice and Note variables that we defined in the Data Object field are received, we ensured that the value of the ticketPrice variable was left blank and the error message string was set into the Note variable. You can watch our other videos to learn the detailed usage of the Script Task activity.



| Activities > Script Task | |
|---|---|
| Name | Script task: Could not be loaded |
| Script format | javascript |
| Script | execution.getVariable("ticketPrice"); execution.getVariable("Note"); <br><br> ticketPrice = ""; Note= "No matching ticket is found."; <br><br> execution.setVariable("ticketPrice",ticketPrice); execution.setVariable("Note",Note); |

- After this step, we closed the website that we opened with the Close activity. For this, in the Close activity, it is enough to select the reference name of the application that we want to close from the list. Finally, we ended the process with an End event.
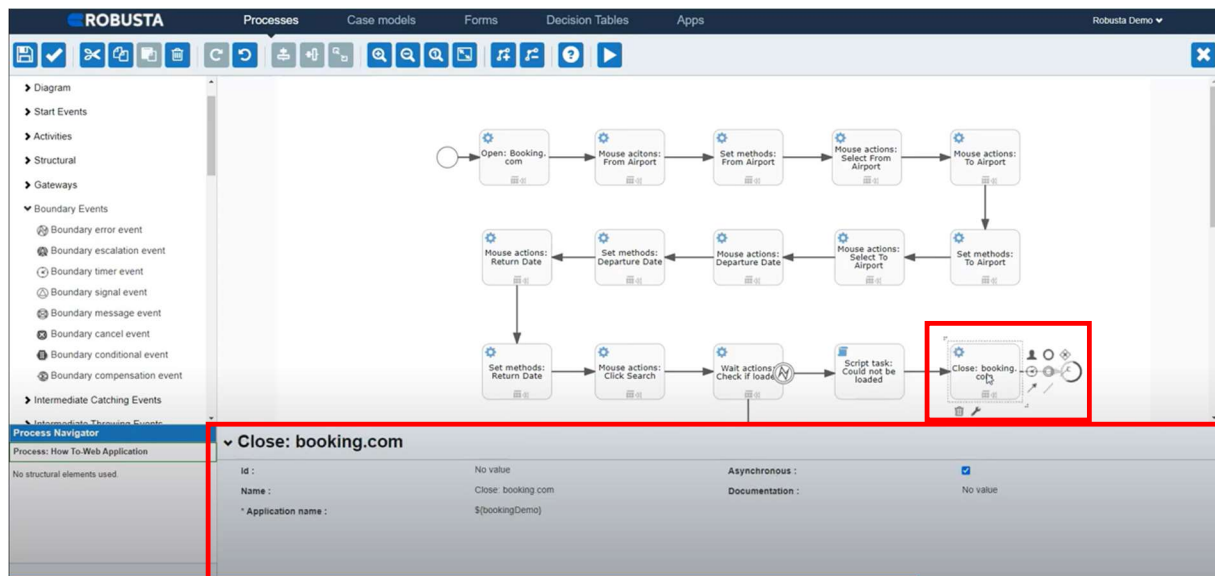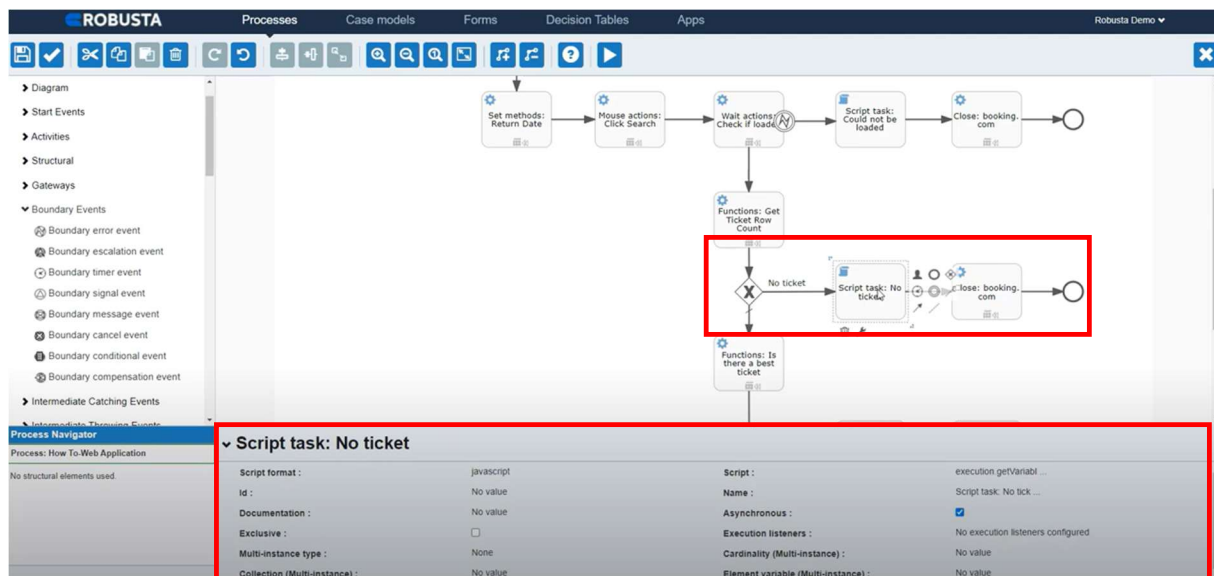


| Applications  > Close | |
|---|---|
| Name | Close: booking.com |
| *Application name | ${bookingDemo} |

- We continued our process with the Function activity to find out how many tickets were on the page when the search was concluded. The Functions activity allows us to get various information of the elements on the page. We entered the location information of the ticket fields listed in this activity in the html structure into the field and chose the element count option from the Action field. Then we wrote the variable we want the results to be assigned to in the Result variable( "*ticketRowCount*" ) name field.
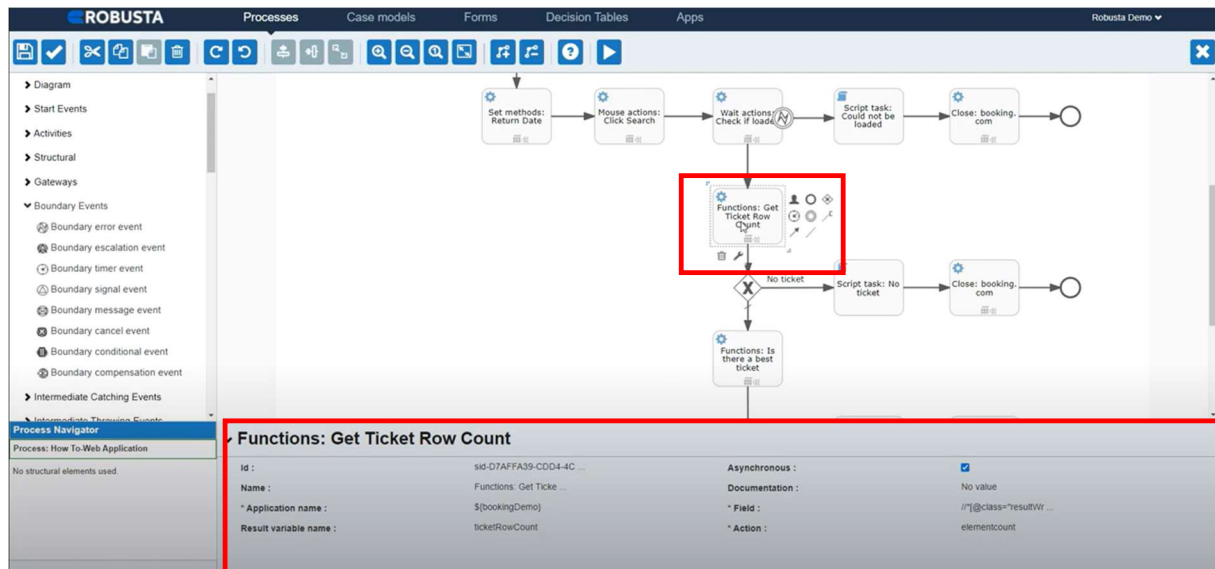


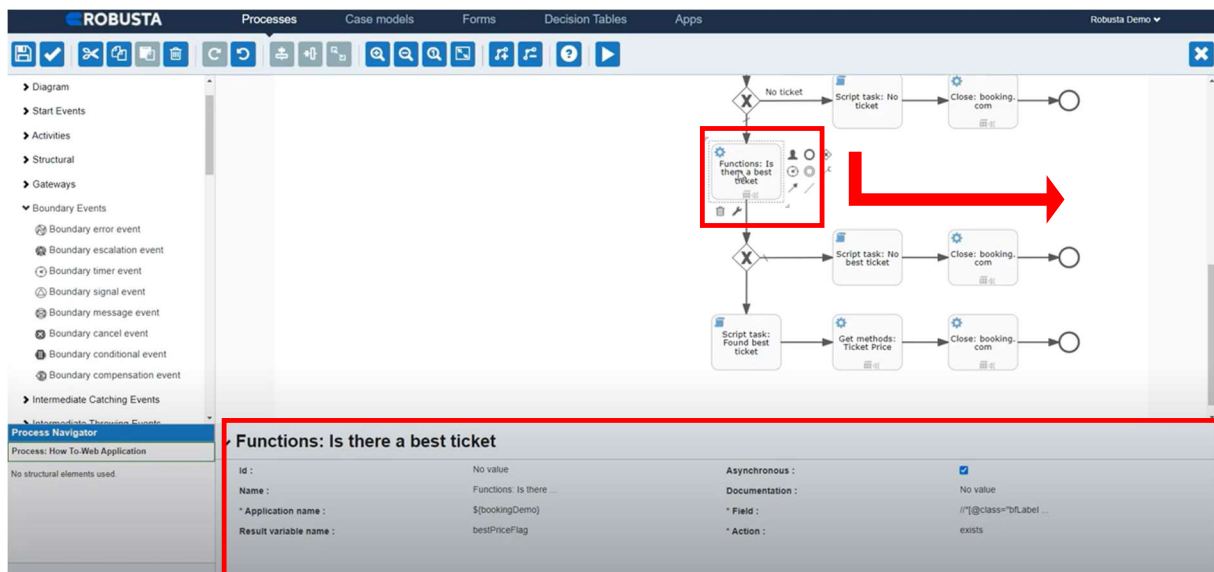| Applications  > Functions | |
| --- | --- |
| Name | Functions: Get Ticket Row Count |
| *Application name | ${bookingDemo} |
| *Field | //*[@class="resultWrapper"] |
| Result variable name | ticketRowCount |
| Action | elementcount |

- Then, according to the variable return from the Function activity, we ensured that if no flight was found, the process was terminated, and if there was a flight, the process continued. To do this, we used a gateway that allowed us to advance our process according to the conditions we set.The flow condition (${ticketRowCount==0}) parameter of the Gateway we used in this process will come from the variable we determined in the previous step. If the value of this variable is 0, that is, there is no ticket, we have ensured that the condition returns true and the process proceeds and ends from here. Otherwise, we chose the default flow option and did not enter any conditions. In this way, the process will continue from here if it does not match a condition.



| Activities > Script Task | |
|---|---|
| Name | Script task: Could not be loaded |
| Script format | javascript |
| Script | execution.getVariable("ticketPrice");<br>execution.getVariable("Note");<br><br>ticketPrice = "";<br>Note= "No matching ticket is found.";<br><br>execution.setVariable("ticketPrice",ticketPrice);<br>execution.setVariable("Note",Note); |

- In the case of there is a flight, we used the Function activity, similar to the previous one, to query whether there is a flight with the best price information on the page. According to the variable return from this activity, if there is no flight with the best price (${bestPriceFlag==false}) information, we finished the process by sending the message No best ticket and closing the page we opened.



| Applications > Functions | |
|---|---|
| Name | Functions: Is there a best ticket |
| *Application name | ${bookingDemo} |
| *Field | //*[@class="bfLabel bf-best"] |
| Result variable name | bestPriceFlag |
| Action | exist |

| Activities > Script Task | |
|---|---|
| Name | Script task: No best ticket |
| Script format | javascript |
| Script | execution.getVariable("ticketPrice"); execution.getVariable("Note"); ticketPrice = ""; Note= "No matching best ticket is found."; execution.setVariable("ticketPrice",ticketPrice); execution.setVariable("Note",Note); |

- After ensuring that an information message is sent if the best ticket is available(${bestPriceFlag==true}), we ensured that the best ticket price is received with the Get methods activity. In this activity, after selecting text from the Type field, we entered the location information of the best price in the field and assigned the result to the ticketPrice variable. Finally, we closed the page and ended our process.
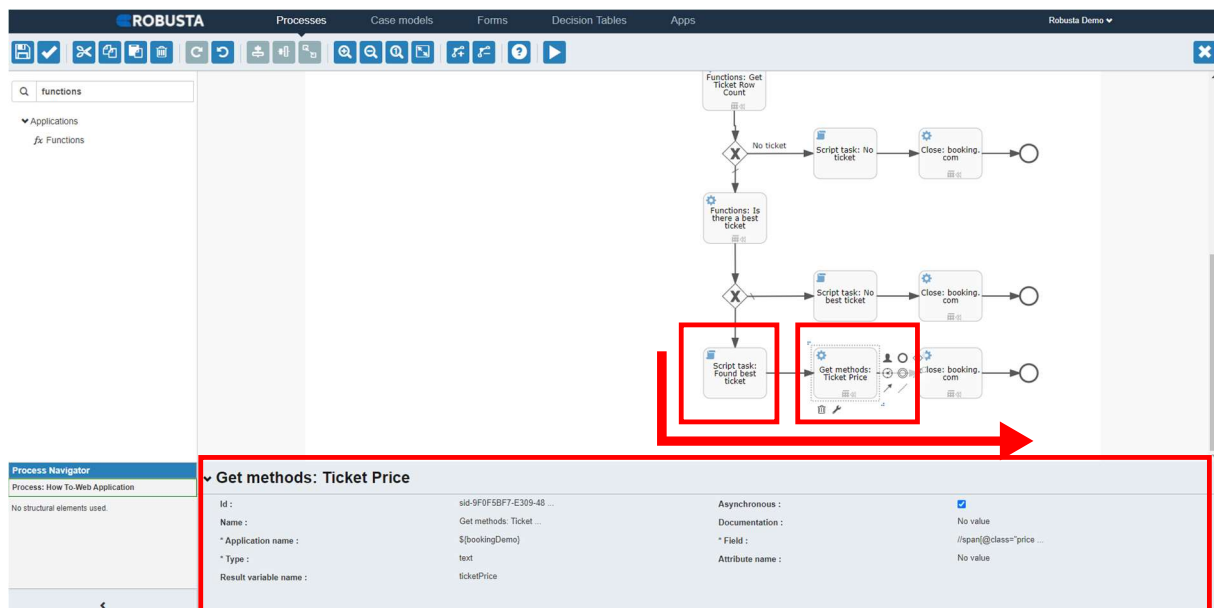


| Activities  > Script Task | |
|---|---|
| Name | Script task: Found best ticket |
| Script format | javascript |
| Script | execution.getVariable("Note"); <br><br> Note = "Best ticket is found."; <br><br> execution.setVariable("Note",Note); |

| Applications   > Functions | |
|---|---|
| Name | Get methods: Ticket Price |
| *Application name | ${bookingDemo} |
| *Field | //span[@class="price option-text"] |
| Result variable name | ticketPrice |
| *Type | text |

- As you can see, the process started to work and the Booking.com website was opened. Required information was entered in the relevant fields and the search button was clicked. The results are loaded and the process has completed. Now let's look at the steps of our process and the variables in the process.

- As you can see on this screen, our process has been ended successfully. You can see the steps of the process by clicking the Show diagram section. Here, you can see the best price information and other variables that occur during the process. We have completed our How-to tutorial. Hope to it was useful for you.We hope to see you next time again.