Writing Data to a REST API (POST and PUT method

Table of Contents

knowledge@outsystems.com

www.outsystems.com

Introduction

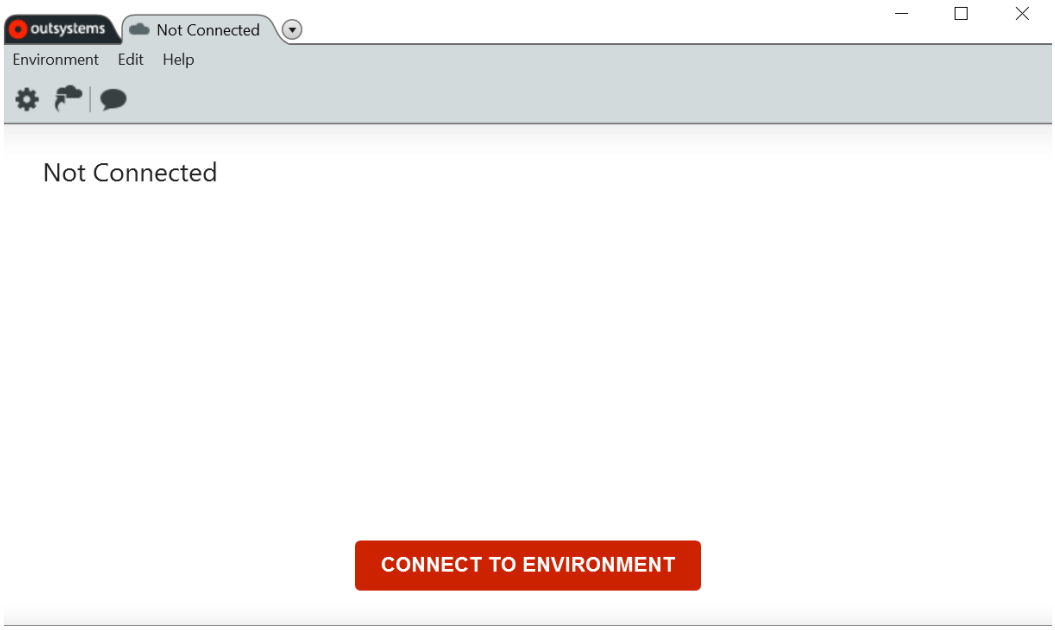
In this lab, we are going to integrate with a REST Web Service to allow to manipulate data in an external system. We are going to use the POST and PUT methods to insert and update information of the Contacts in the external system.

Connect to an Environment

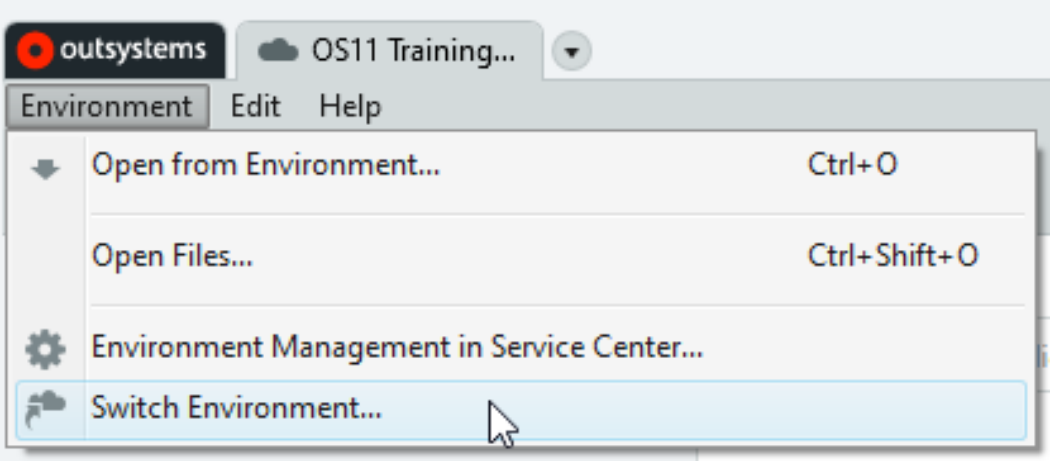
When we open Service Studio for the first time, we will need to connect to an **environment** where the OutSystems platform server generates, optimizes, compiles, and deploys OutSystems applications.

1) Open Service Studio and access the **Connect to Environment** dialog. This can be done in two ways.

   a) If you are not logged in to any environment, click on **Connect to Environment**.



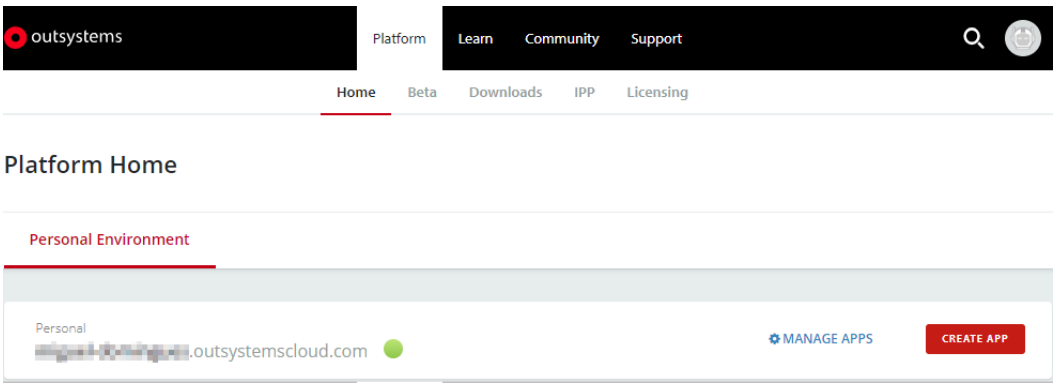
b) If you are already logged in to an environment, select the **Switch Environment...** option from the Environment menu at the top.



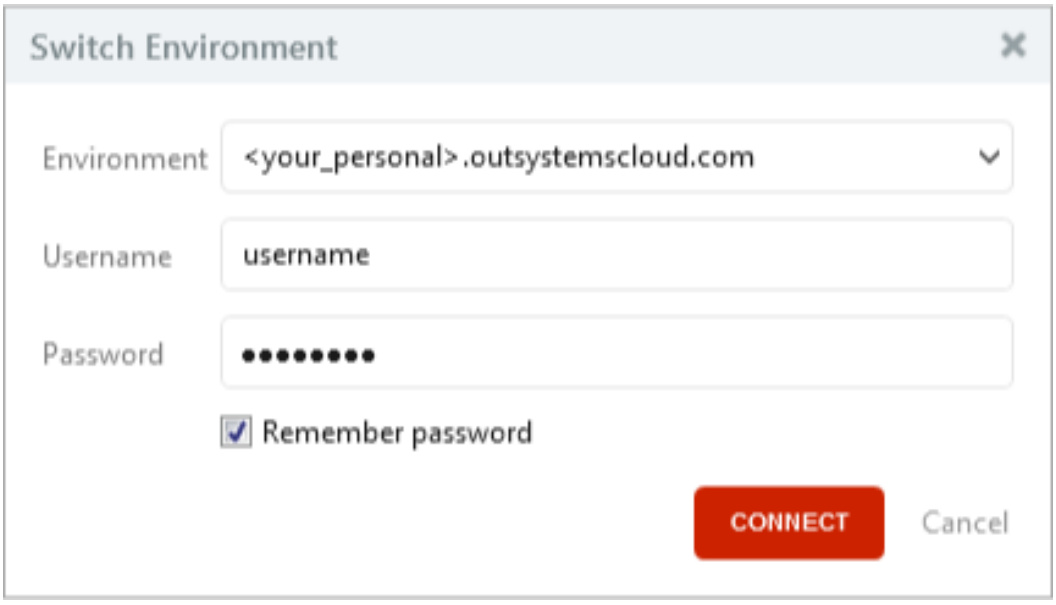
2) Connect to your OutSystems personal environment.

a) If you are using your Personal Environment, you can find its address in the OutSystems website and log in.

b) Under the **Platform** tab and then under the **Personal Environment** tab the environment address (or **Server Address**) can be found.



c) Back in Service Studio, use that Environment and login with your OutSystems community email (username) and password.



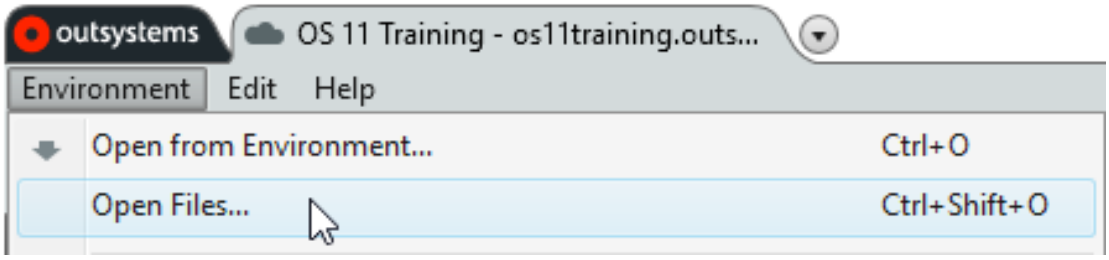
Get to know the scenario

Install the Contacts application

If you have completed the **Reading Data from a REST Web Service (GET)** exercise lab before you can skip this section and jump to the next page. Otherwise, follow the instructions below to install the quick start application.

Open and publish the **REST Contacts - POST and PUT.oap** in your personal environment. The oap file can be found in the Resources folder.

1) In the Applications, open the Environment menu and select **Open Files...**



2) In the Open dialog, change the File Type dropdown option to **OutSystems Application Pack (*.oap)** and then open the REST Contacts - POST and PUT.oap.

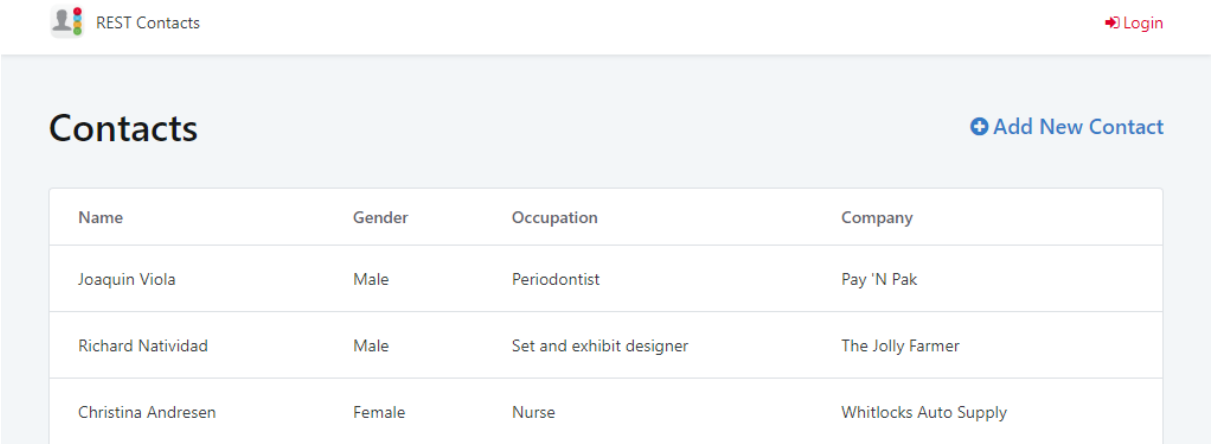
3) Click Proceed when asked.

4) Wait for the installation to complete and then proceed.

Business Case Overview

The REST Contacts application is a simple application, only containing a couple of Screens, Preparations, and some Screen Actions. We already list Employees on one of the screens. Now, we'll create the logic to create a Contact or update an existing one.

The quick start application is simply to speed up the setup part and start right away working with the external REST web service.



Consume a POST REST API Method

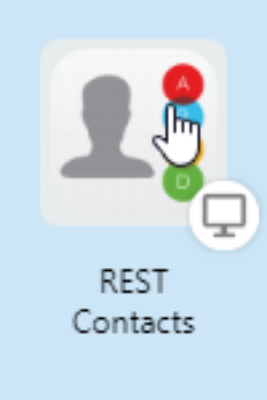
In OutSystems, integration with REST Services can be straightforward. OutSystems helps us to generate all the methods and data structures needed to integrate with an external system.

Before you consume any REST API it's important to gather all the information you need from the REST API documentation. Information such as the expected structures and some examples may be be really useful when consuming an external service. In this lab, the documentation for the external REST Web Service is available here.

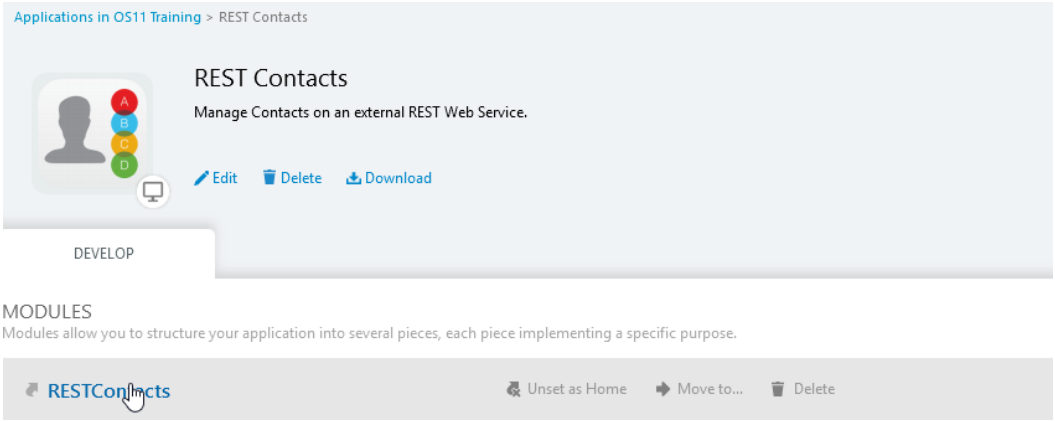
The POST request is known as a method to insert new data into the external resource when calling. In this section we will extend the integration with the external REST Web Service. The extended integration will allow to create data on the external web service thru a form provided in our application.

1) Open the Contacts module

   a) In the Applications list, locate the **REST Contacts** Web application and open it.

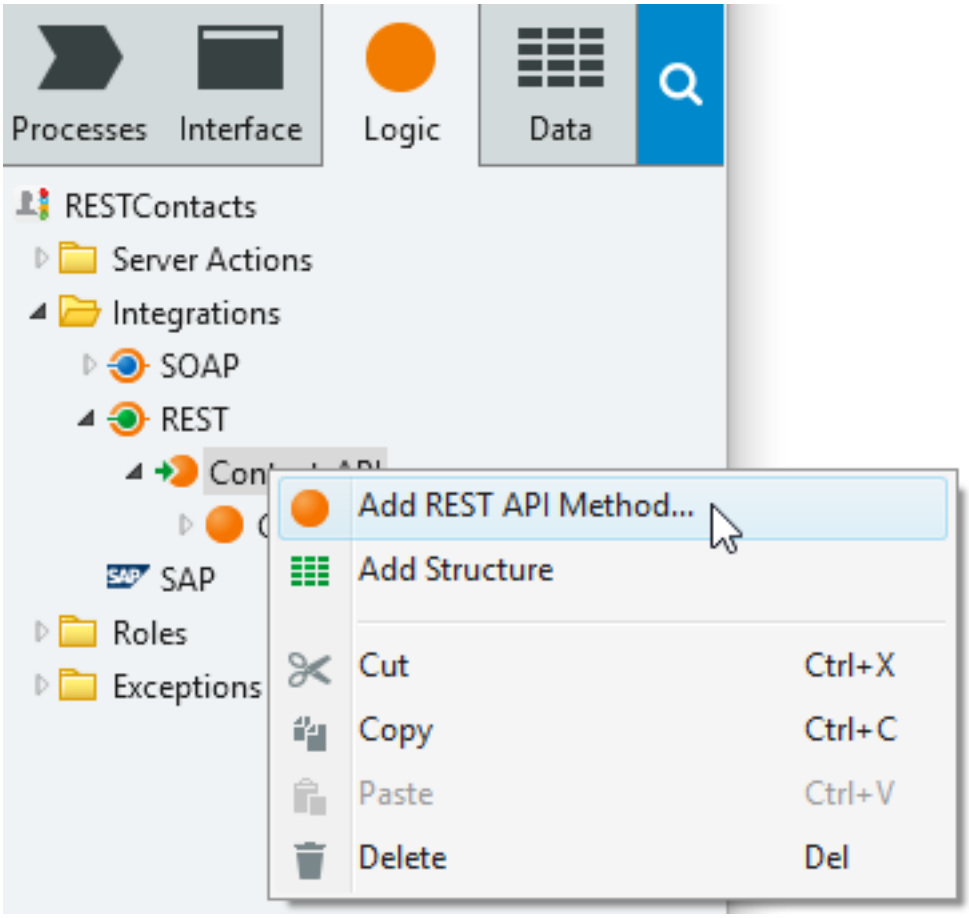


   b) Open the **RESTContacts** module



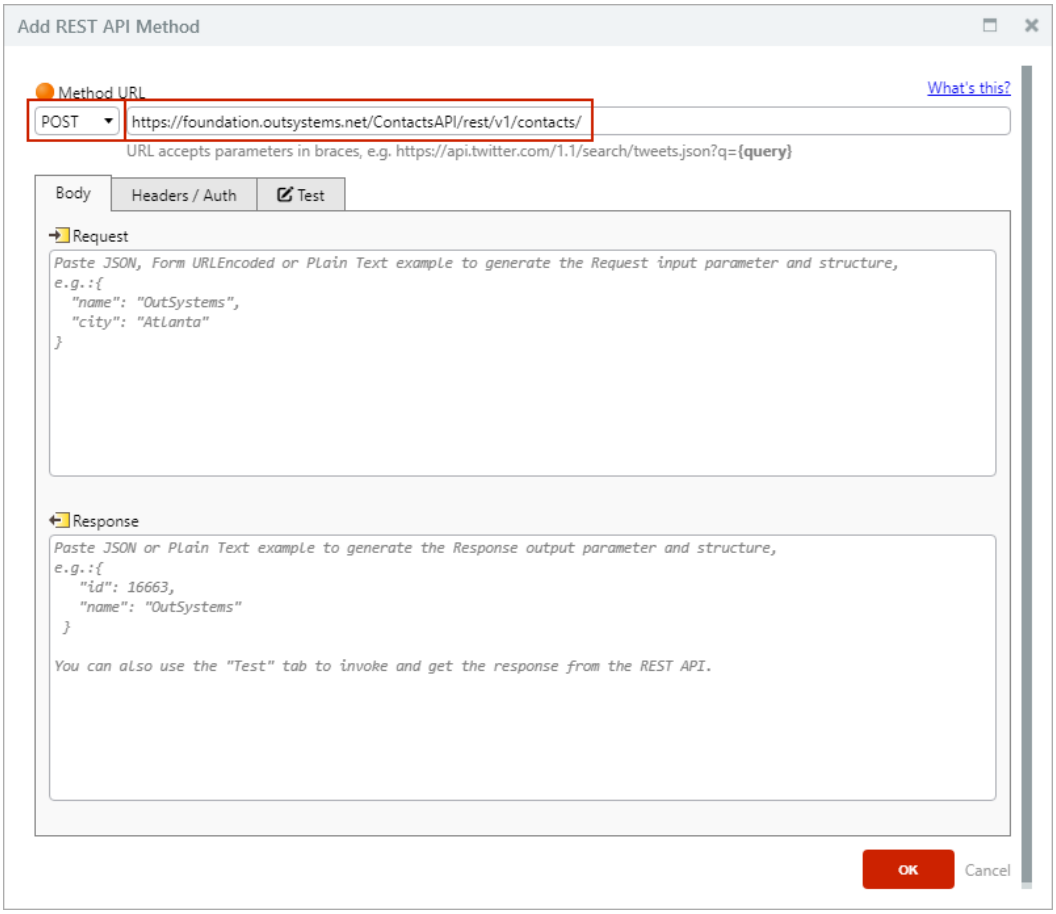
2) Consume the POST method of the external REST Web Service.

   a) Switch to the Logic tab and in the Integrations folder, right-click the **REST** element and select *Add REST API Method...*. This opens a window to configure the REST method that you want to consume.



   b) Set the Method to *POST* and the URL to

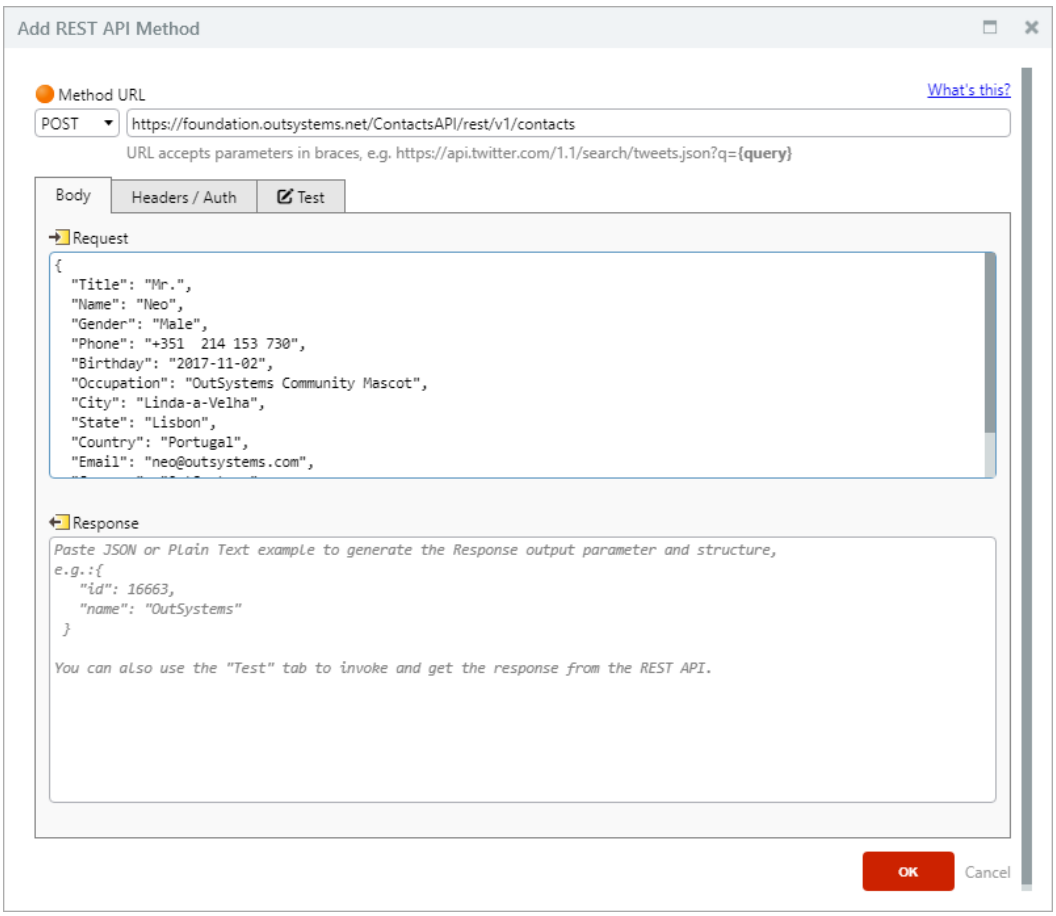
```
https://foundation.outsystems.net/ContactsAPI/rest/v1/contacts/
```



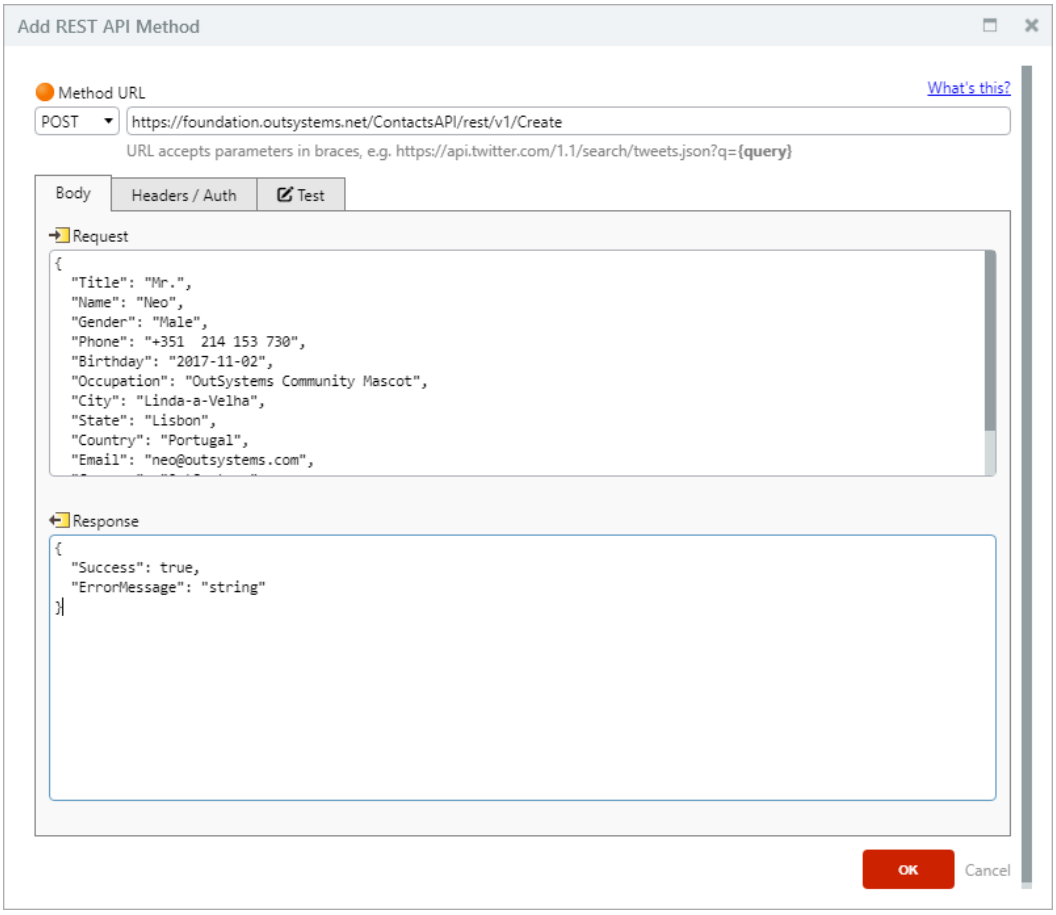
c) Since this API Method has an input, this is where Documentation is handy and will allow you to move forward. In this case the expected input is:

```
{
"Title": "Mr.",
"Name": "Neo",
"Gender": "Male",
"Phone": "+351 214 153 730",
"Birthday": "2017-11-02",
"Occupation": "OutSystems Community Mascot",
"City": "Linda-a-Velha",
"State": "Lisbon",
"Country": "Portugal",
"Email": "neo@outsystems.com",
"Company": "OutSystems"
}
```

d) Copy the above sample input to the **Request** text area



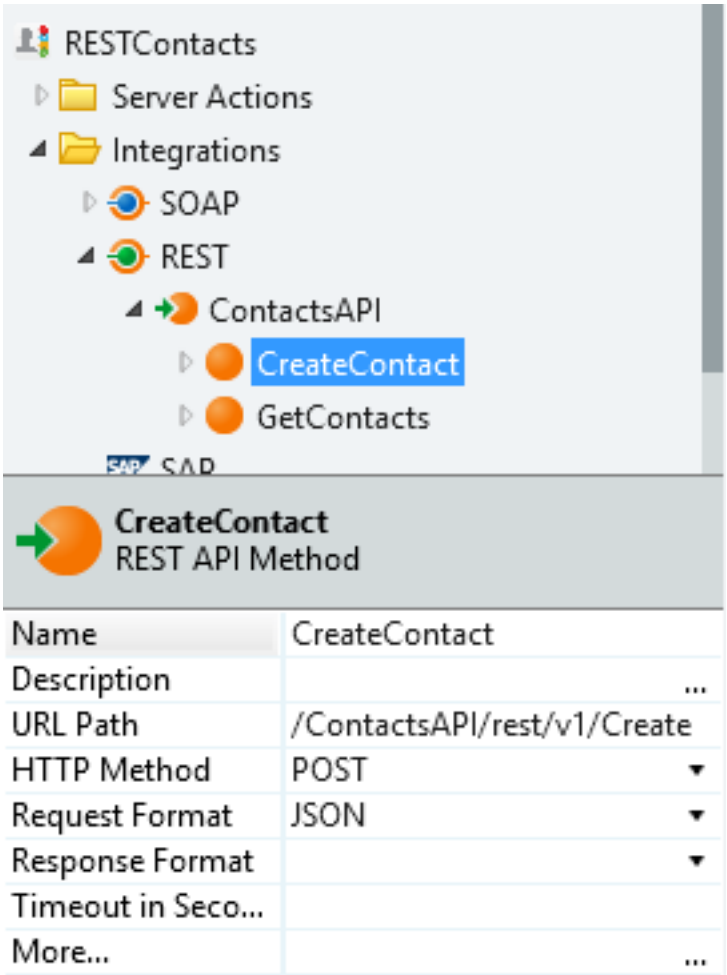
e) From the API Documentation we can extract the sample response. Copy the sample response and place it on the **Response** text area.



NOTE: You can also Test the invocation (under the Test tab) and obtain the response that can be then copied to the Body tab. This will allow Service Studio to infer the response data type structure. Notice that using the Test option on methods that modify data on the external system, will actually change the data, and it may be undesirable.

f) Click the **OK** button to close the Add REST API Method.

g) Finally, change the name of the method from PostContacts to CreateContact.



NOTE: This step is not mandatory, but it allows developers to have custom or more familiar names in their integrations.

Add Contact User Interface

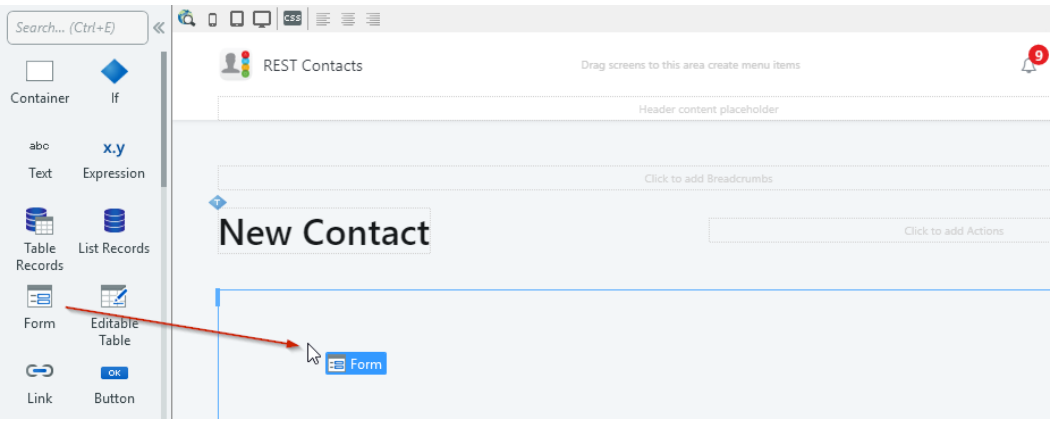
In this section we will modify the existing (empty) screen ContactDetail to allow to create a new contact. In our sample app, the ContactDetail screen has already been created. It has an input parameter (named Email) that will be later used to edit contacts. From the main screen (Contacts) a link already exists to add a new contact that redirects the end-user to the ContactDetail screen.

Let's now implement the ContactDetail User Interface and the Save Screen Action to create a new contact in the external system.

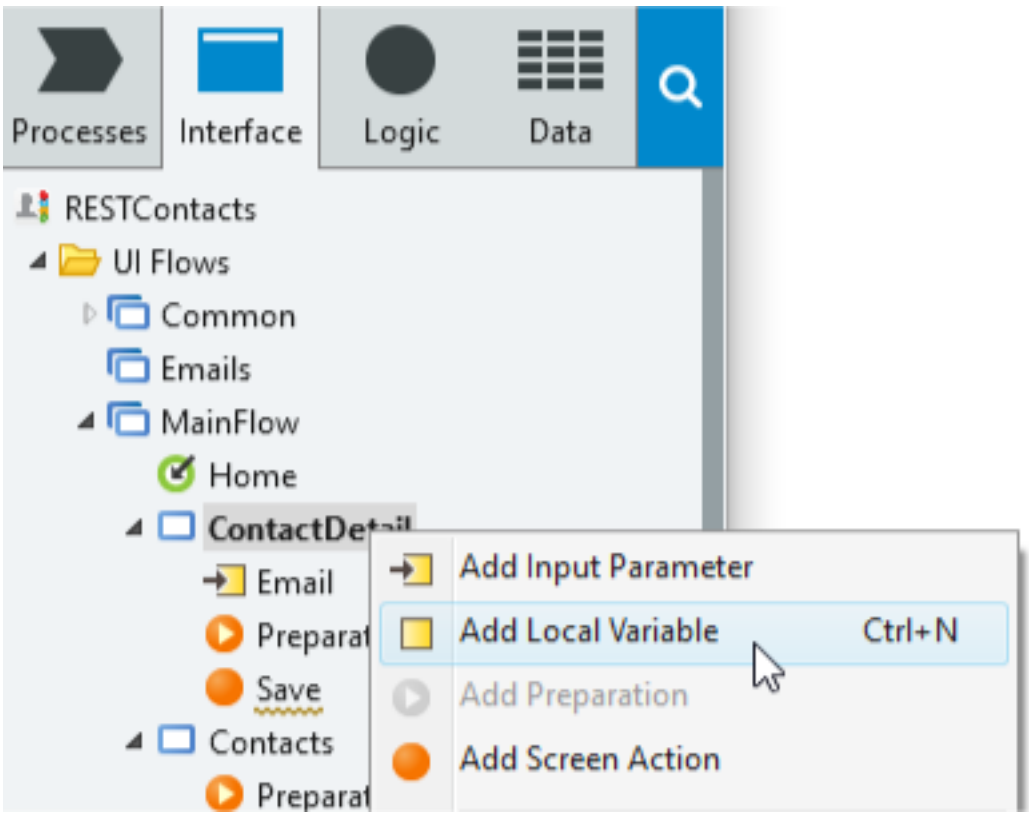
1) Create a new Form in the ContactDetail screen with the required inputs for the CreateContact REST API Method. Define the logic to invoke the REST API Method inside the Save Screen Action of the same screen.

   a) In the Interface tab, open the **ContactDetail** Web Screen.

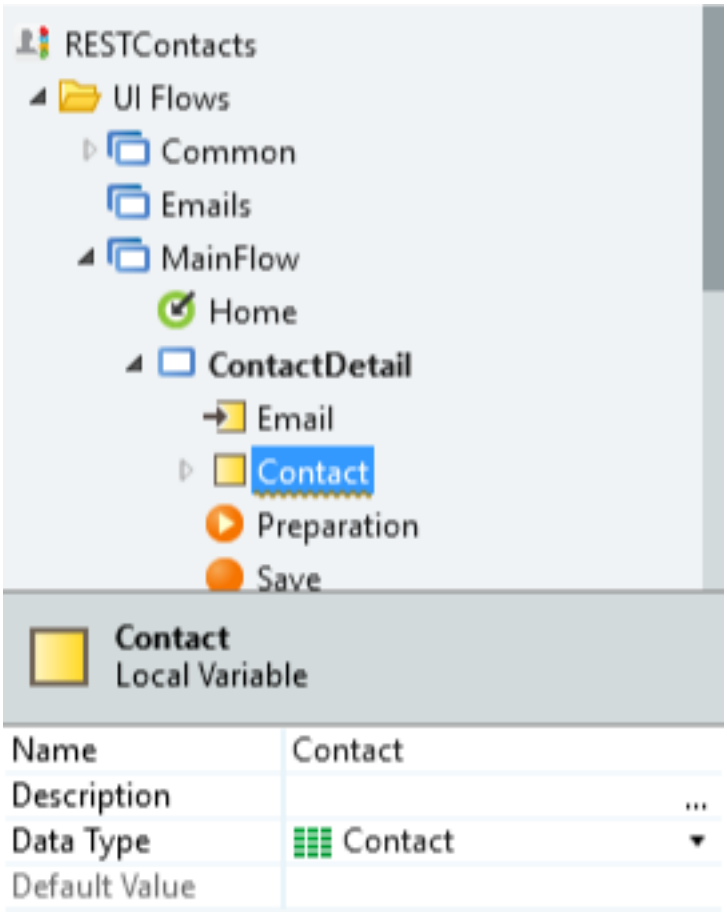
   b) Drag a **Form** widget and drop it in the Main Content area of the screen.



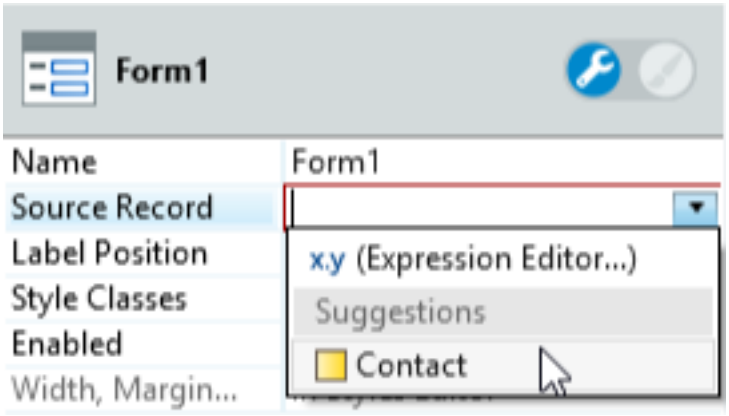
c) Right-click on the **ContactDetail** screen and select *Add Local Variable*



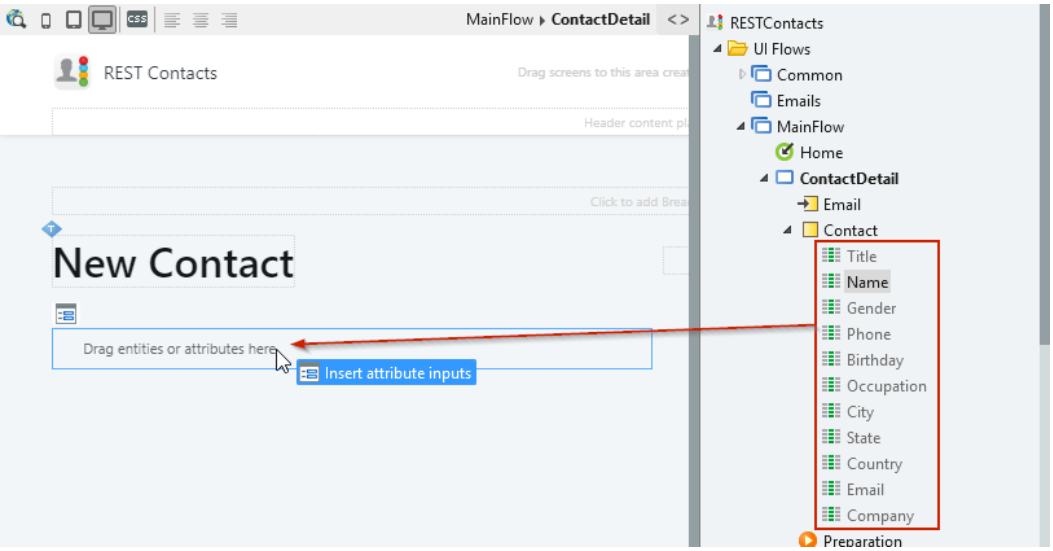
d) Set the Variable local variable **Name** to Contact, and make sure the **Data Type** is set to *Contact*.



e) Select the Form in the screen preview and then set the **Source Record** property to the *Contact* local variable.

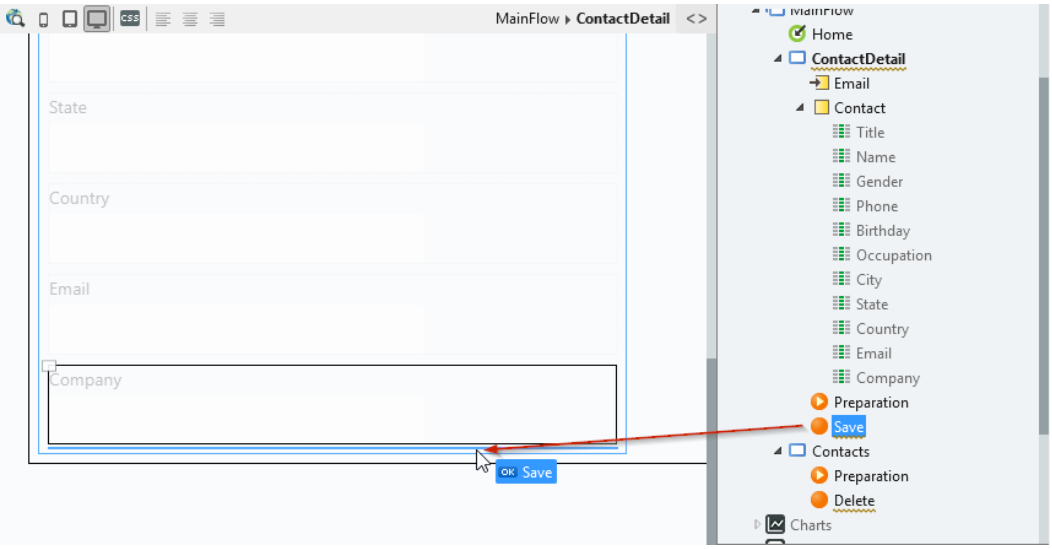


f) Expand the Contact local variable, **select all attributes** and drag and drop them to the form area.



NOTE: This drag and drop action will create the Label and input for each of the attributes. Each input will match the actual Data Type of the attribute. For instance, the Birthday input will display a Calendar in runtime. We won't go further in terms of customizing the form, and will focus on actually creating a Contact using the REST API.

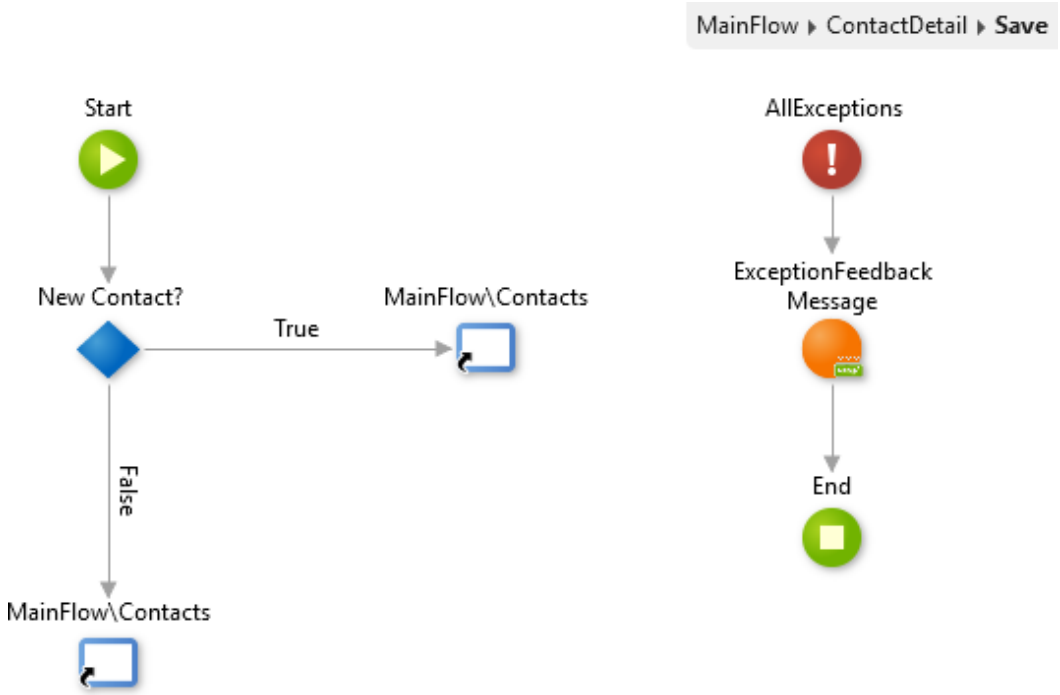
g) Drag the **Save** screen action from under the *ContactDetail* screen and drop it at the bottom of the Form



NOTE: Dragging a Screen Action into the Screen will automatically create a Button, set its label to the Screen Action name and its Destination to the same Screen Action.

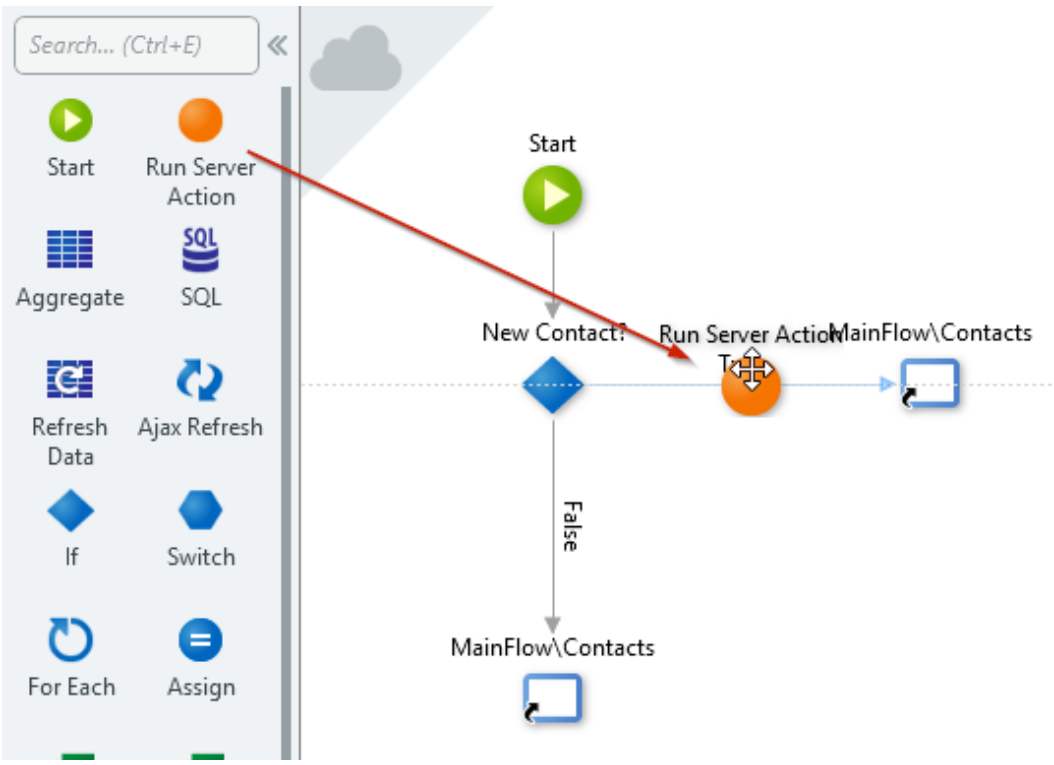
2) Create the logic inside the Save screen action that will invoke the REST API to create a new contact based on the input entered by the user in the Form.

   a) Double-click the button to open the **Save** screen action, or open it from the elements tree (under the *ContactDetail* screen)

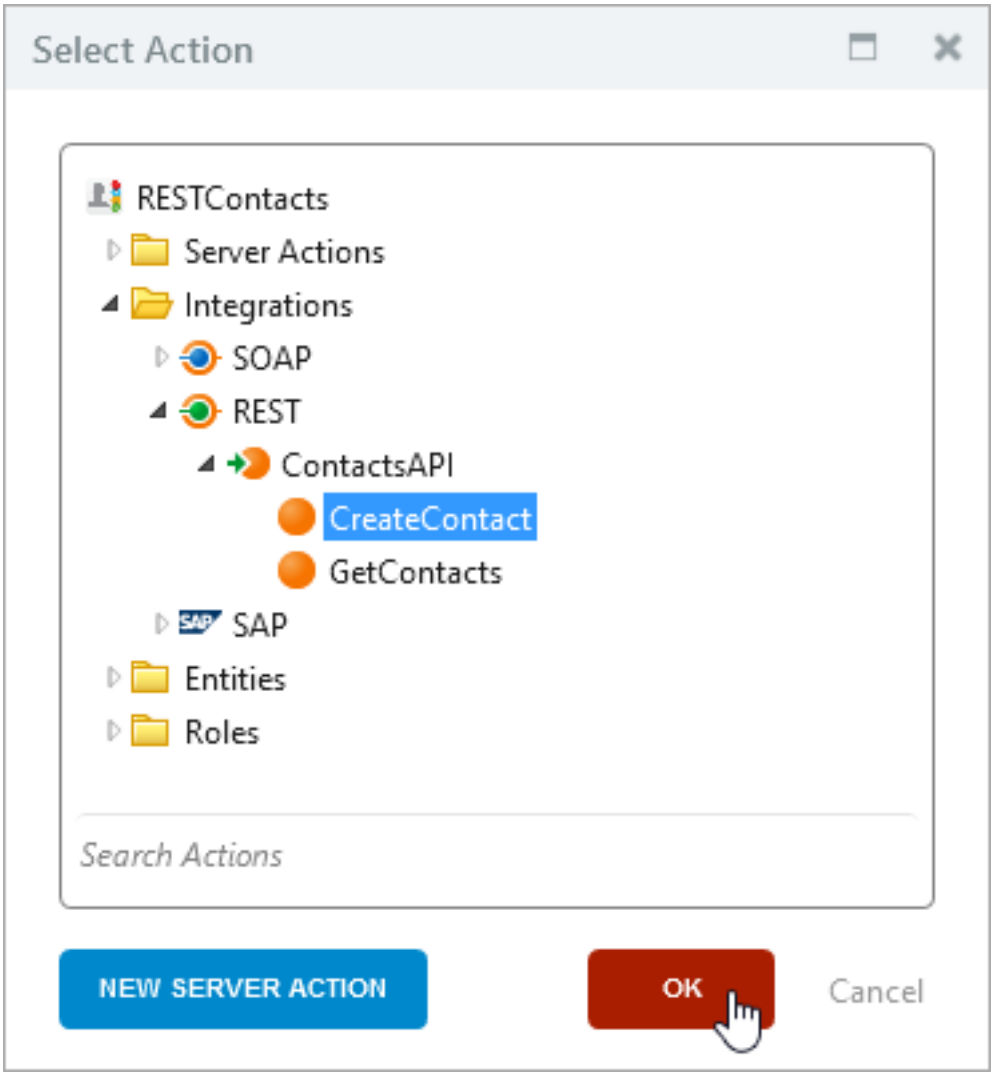


NOTE: This action already has some logic defined. The New Contact? If verifies if the Email input parameter of the screen is empty or not. When empty, it means that we will be adding a new contact. Later on, we will implement the logic regarding updating existing contacts.

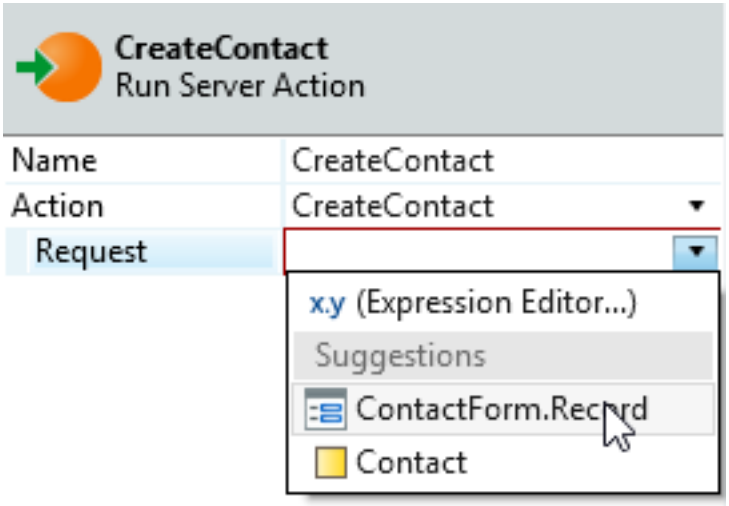
b) Drag a **Run Server Action** and drop it on the True branch



c) In the **Select Action** dialog, select the CreateContact action under the ContactsAPI REST integration.



d) Set the **Request** parameter to ContactForm.Record



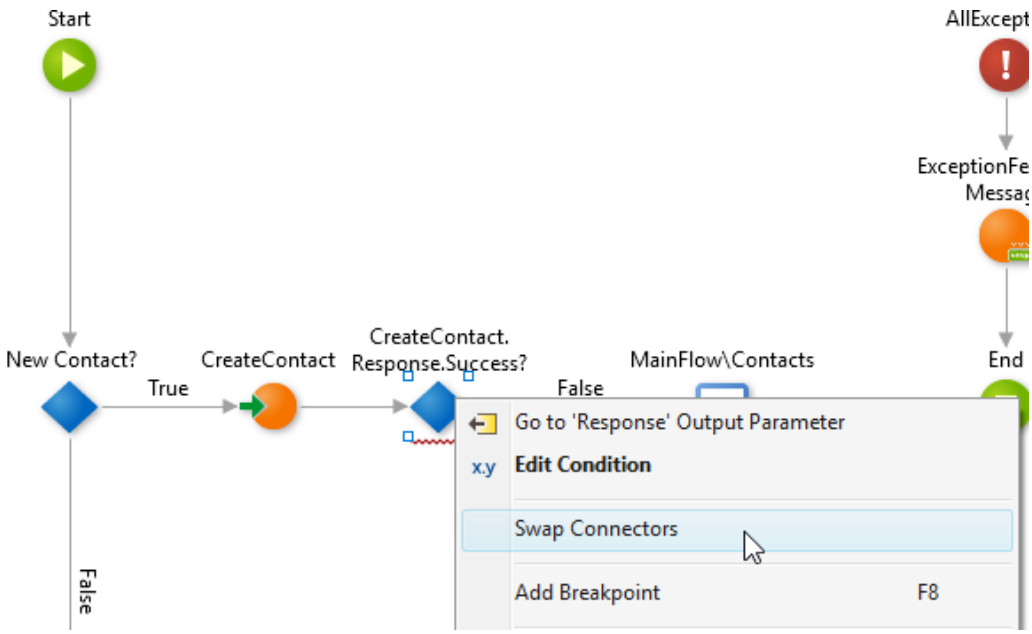
3) Validate if the Contact was created successfully. If not, show a feedback message containing the error message.

   a) Drag an **If** and drop it between the *CreateContact* and the *Contacts*.

   b) Set the **Condition** of the If to

```
CreateContact.Response.Success
```

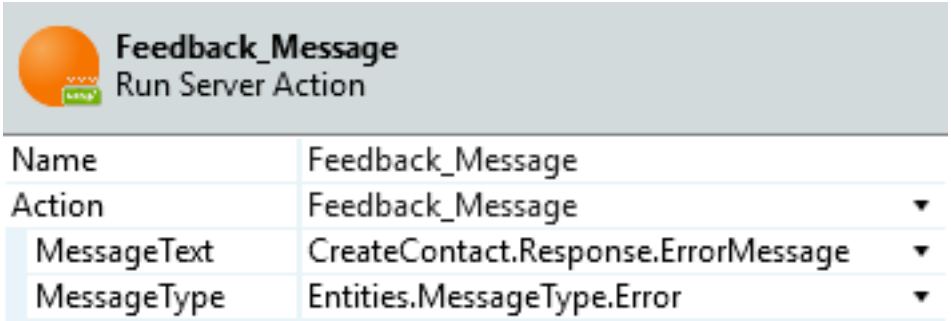
   c) Right-click the **If** and select *Swap Connectors*



   d) Drag a **Run Server Action** and drop it below the existing If, then select the *Feedback_Message*.

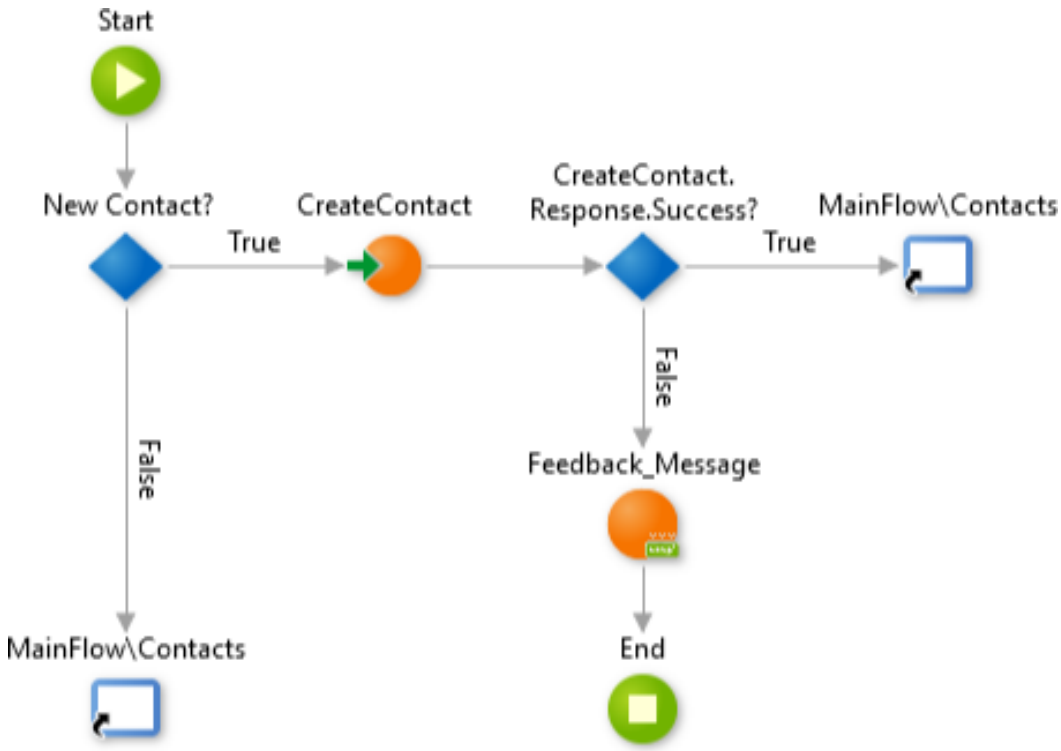
   e) Create the False branch connector from the *If* to the *Feedback_Message*.

   f) Set the parameters of the **Feedback_Message** as follows



g) Drag an **End** and drop it below the *Feedback_Message* element, and then create the connector between both.

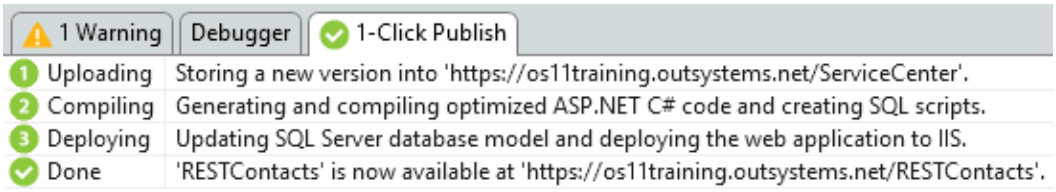
h) The flow should look like this



4) Publish the application using 1-Click Publish button and verify that the publish completed successfully in the 1-Click Publish Tab.

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

b) Verify in the 1-Click Publish tab that the publishing process was successful.



c) Preview the app in the browser by clicking on the **Open in Browser** button.

d) Click the *Add New Contact* link at the top right of the screen.

e) Fill in the form with some Contact information as, for example:

Title: Mr.

Name: Neo

Gender: Male

Birthday: 2017-11-02

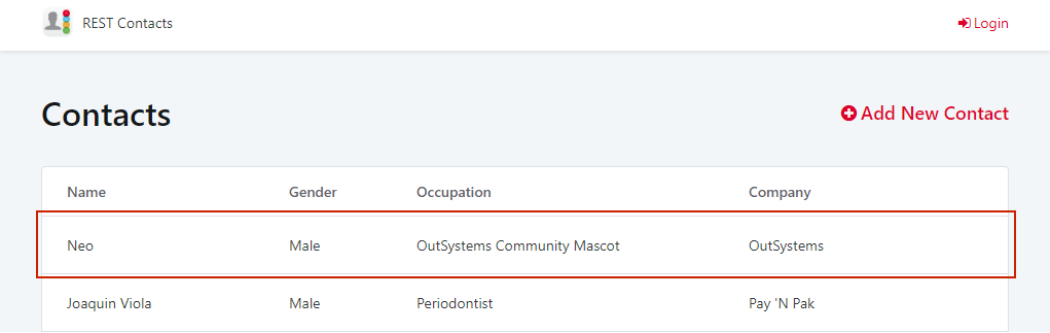
Occupation: OutSystems Community Mascot

Email: *<your email>*

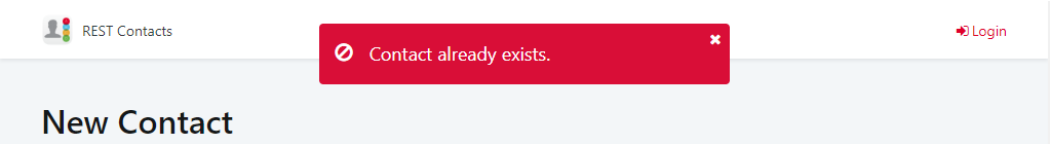
Company: OutSystems

f) Click **Save**.

g) After saving, you should be redirected to the Contacts screen, and see the new contact.



h) Try to enter another contact with the same email. Upon clicking Save, an error message should appear.



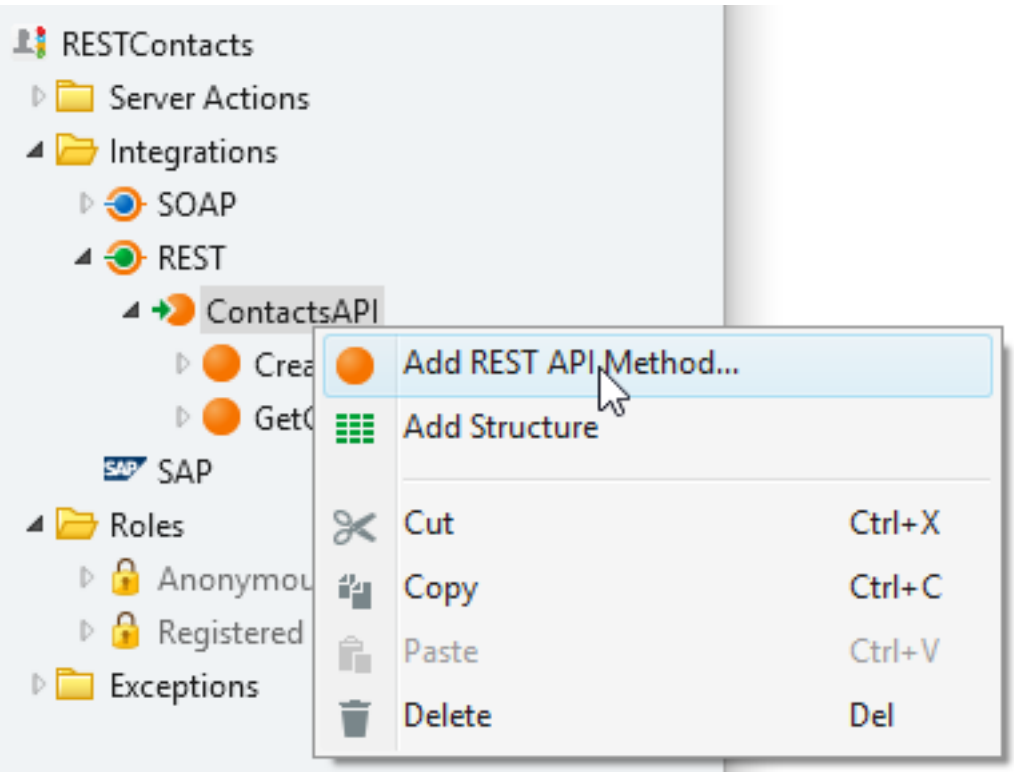
Consume a PUT REST API Method

In this section we will start working towards having the ability to update existing contacts. First we will start by adding a new method to our API that will provide this functionality. Later on, we will modify our user interface and logic to allow users to update existing contacts.

To enable updating contacts an extra GET method will be needed. This one to retrieve a single contact based on a given email address. This will allow to retrieve the contact information in the Preparation of the ContactDetail screen, and then on the Save action update the contact using the PUT method.

1) Consume the PUT REST API Method that given an Email will update the remaining attributes of the contact.

   a) Switch to the Logic tab, right-click the **ContactsAPI** and select *Add REST API Method...*



   b) Set the Method to **PUT** and the *URL* to

```
https://foundation.outsystems.net/ContactsAPI/rest/v1/contacts/
```

   c) Add the following sample to the **Request** text area

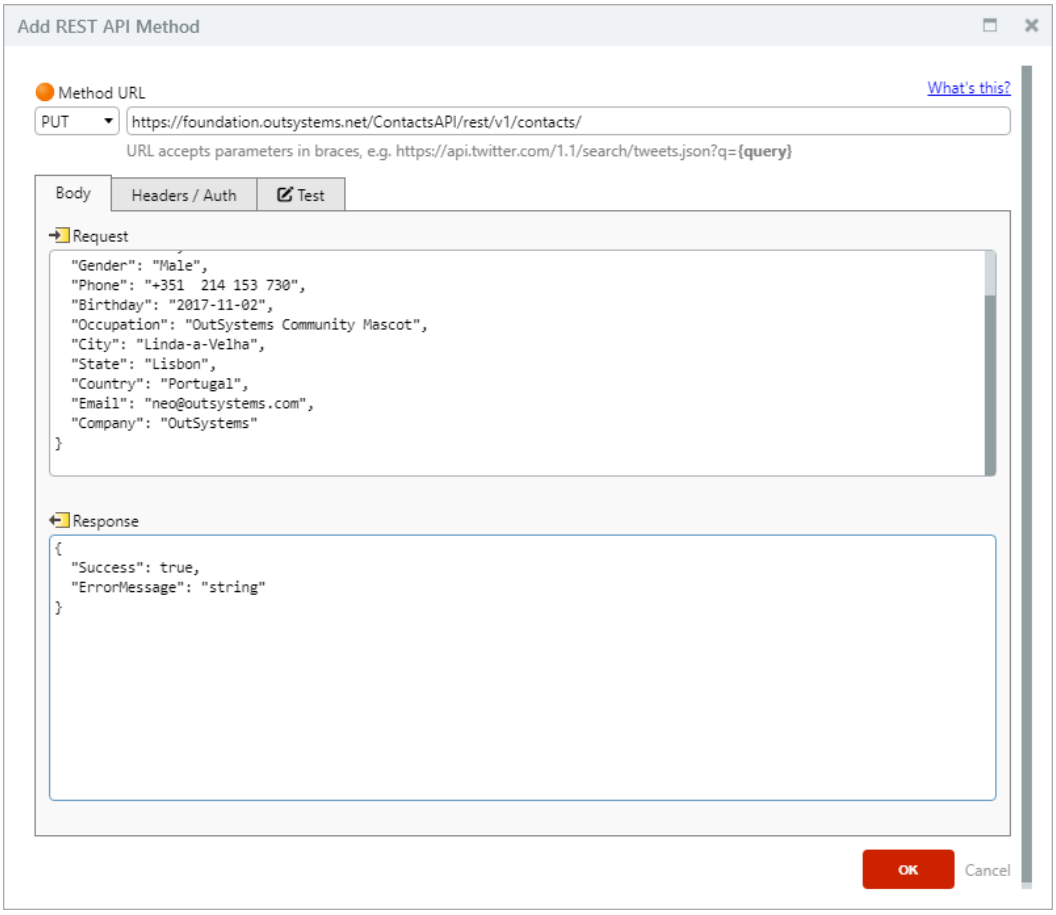
```
{
"Title": "Mr.",
"Name": "Neo",
"Gender": "Male",
"Phone": "+351 214 153 730",
"Birthday": "2017-11-02",
"Occupation": "OutSystems Community Mascot",
"City": "Linda-a-Velha",
"State": "Lisbon",
"Country": "Portugal",
"Email": "neo@outsystems.com",
"Company": "OutSystems"
}
```

NOTE: The sample above has custom values, but the API Documentation also provides a sample input value that you can use.

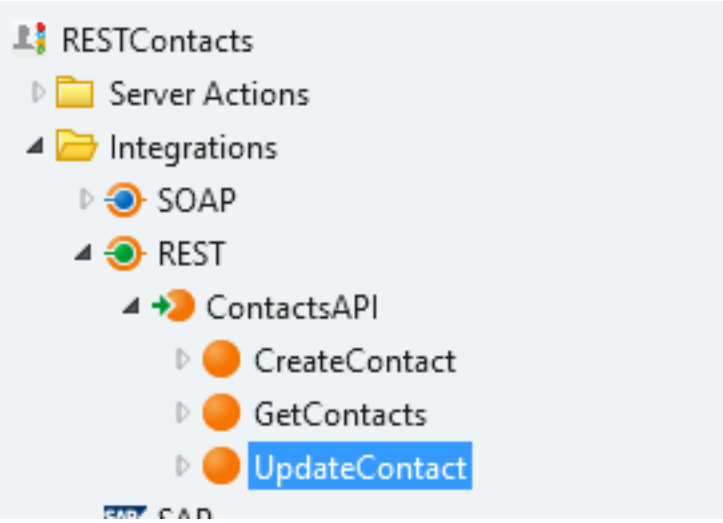
d) Set the **Response** to

```
{
"Success": true,
"ErrorMessage": "string"
}
```

e) Click **OK** to close the dialog.



f) Change the name of the method from *PutContacts* to *UpdateContact*.



2) Consume the GetContact API method to retrieve a single contact based on an Email address.

a) Right-click the **ContactsAPI** and select *Add REST API Method...*.

b) Set the method to GET and the URL to

```
https://foundation.outsystems.net/ContactsAPI/rest/v1/contacts/{Email}/
```

NOTE: The *{Email}* part at the end of the URL defines an input parameter. This syntax informs Service Studio to create a parameter named Email. This is another way of passing information when calling an external REST service. In this case the value is sent as an URL parameter. With the GET method you may also use Headers.

c) Set the Body text are to

```
{
"Result": {
"Success": true,
"ErrorMessage": ""
},
"Contact": {
"Title": "Mr.",
"Name": "Scott Williams",
"Gender": "Male",
"Phone": "+299 91 63 66",
"Birthday": "1988-05-23",
"Occupation": "Support specialist",
"City": "Qasigiannguit",
"State": "Qaasuitsup",
"Country": "Greenland",
"Email": "ScottSWilliams@gustr.com",
"Company": "Dee's Drive-In"
}
}
```


NOTE: You may optionally use the Test tab. In this case it is required to provide an Email to test the method. Note that with methods (e.g. POST, PUT) that change information on the external system, those operations will actually modify the data on the external system. For this reason it is recommended to use different accounts (or endpoints) when developing, and when the application is in Production. Such configurations can be made in Service Center, check here for more information.

d) Click **OK** to close the dialog.

Update Contact User Interface

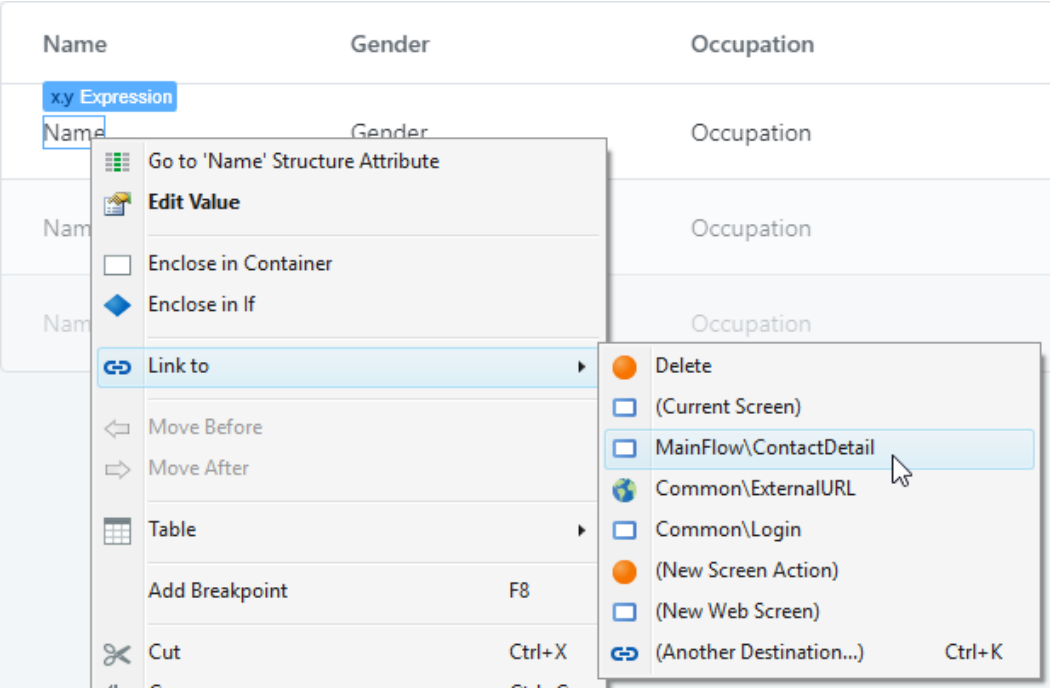
Now that we have all the required integration methods completed, we will change our user interface and logic to allow end users to update contacts.

First we will link the Contacts and ContactDetail screens. Then, on the ContactDetail screen the information of the contact being edited will be fetched from the external system, and upon saving the information, the new data will be sent and stored on the external system.

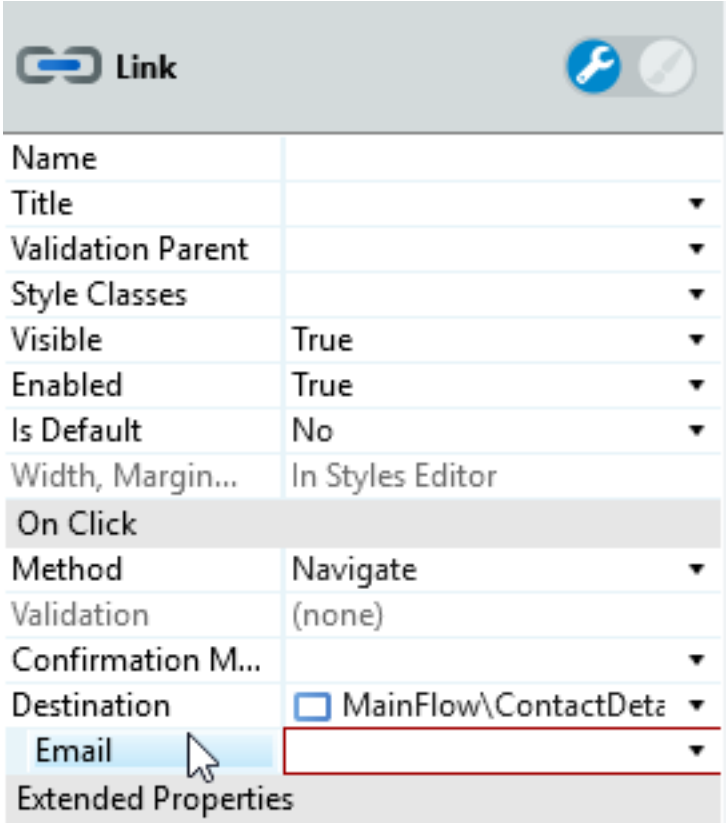
1) Create a link from the Contacts screen to the ContactDetail screen for each row in the Table Records.

   a) Open the **Contacts** screen from the Interface tab.

   b) Select the **Name** expression, right-click it and select *Link to* > *MainFlow\ContactDetail*

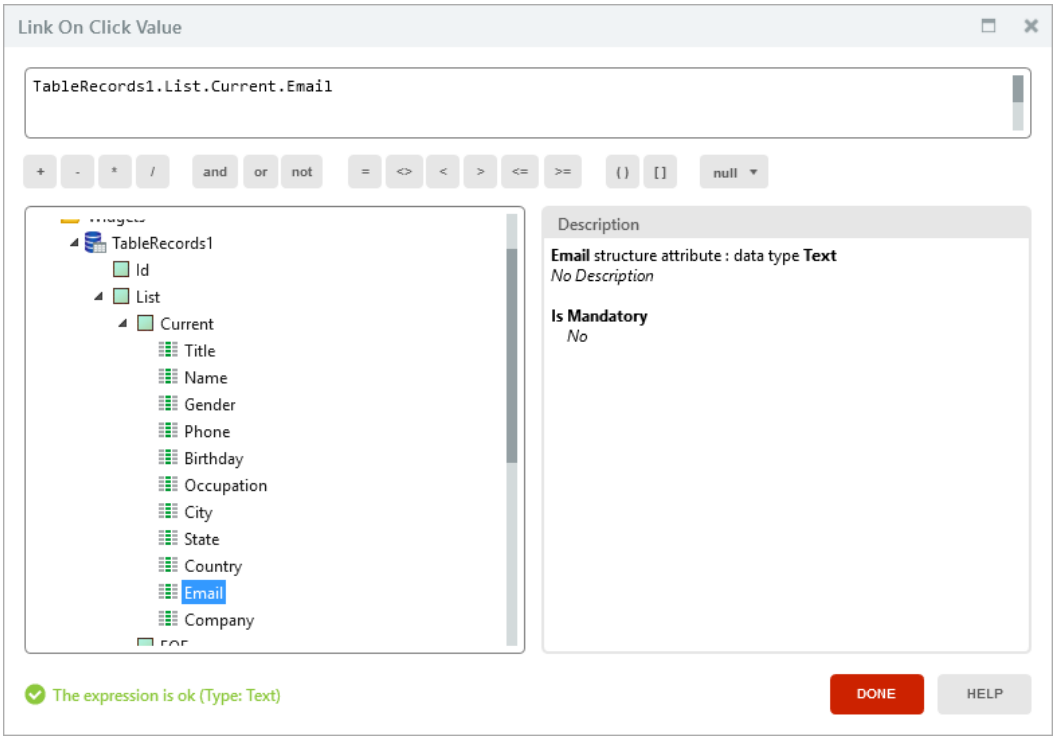


c) Double-click the Email input parameter on the properties pane



d) In the Expression Editor dialog write the following

```
TableRecords1.List.Current.Email
```

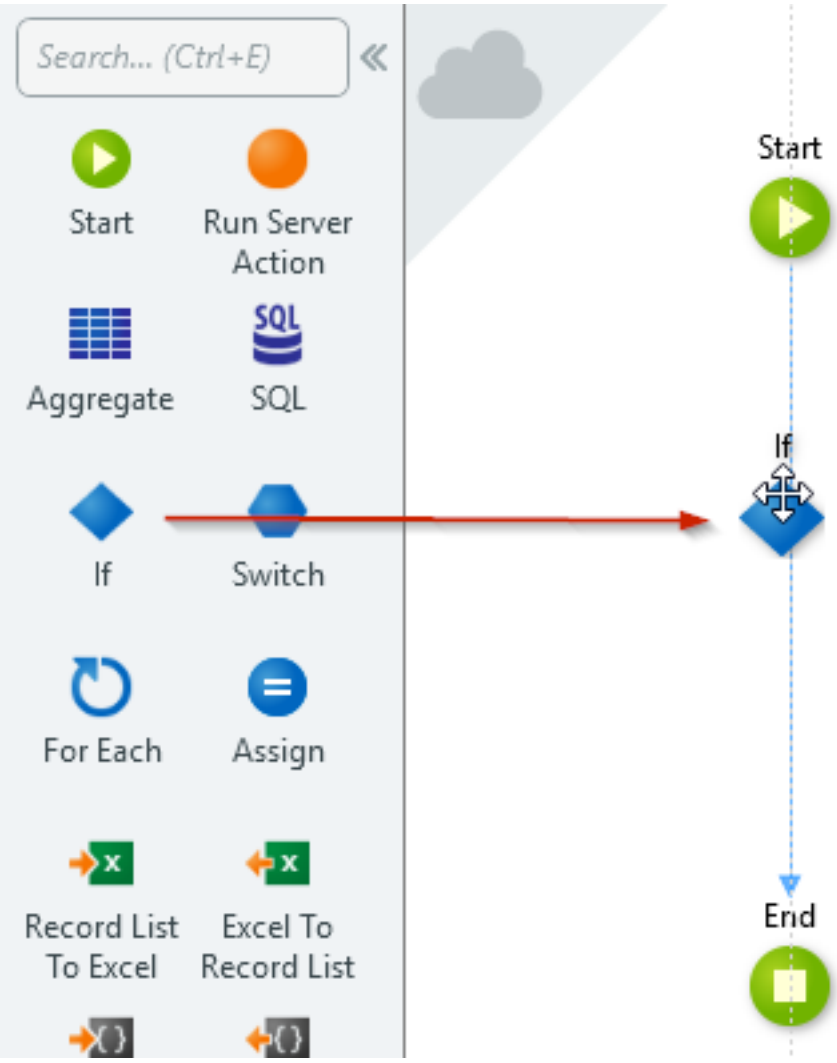


e) Click **Done** to close the Expression Editor.

2) Change the Preparation of the ContactDetail screen to fetch a single Contact from the external system when the Email input parameter is not empty.

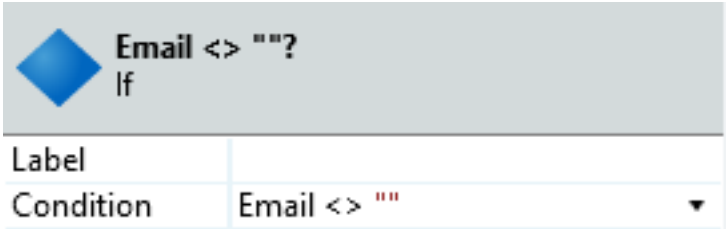
a) Open the Preparation of the **ContactDetail** screen.

b) Drag an **If** and drop it between the Start and End

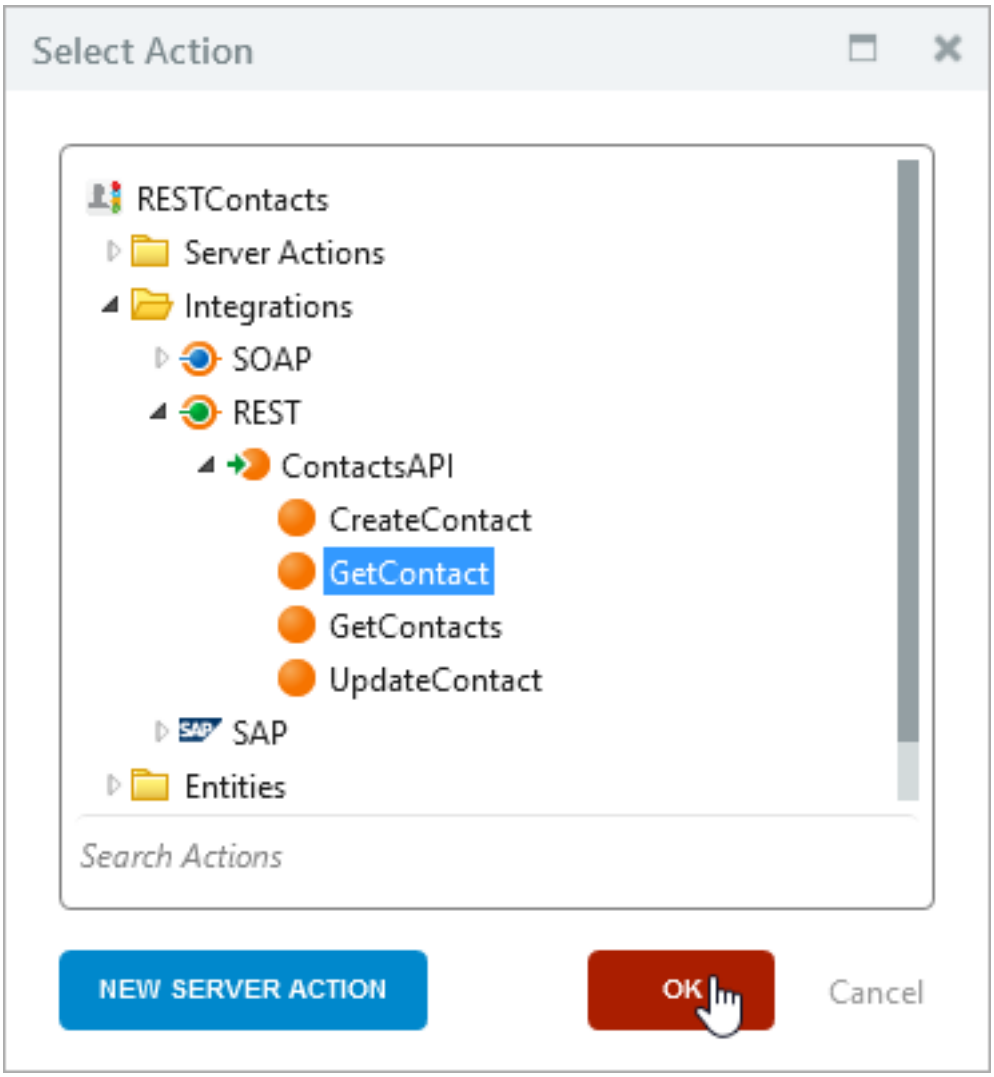


c) Set the Condition property of the If to

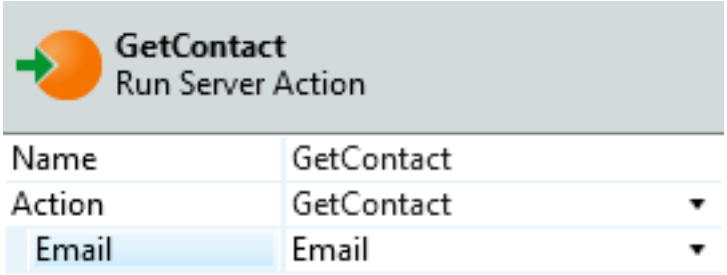
```
Email <> ""
```



d) Drag a **Run Server Action** and drop it to the right of the If, then in the Select Action dialog, choose the GetContact from the ContactsAPI REST integration



e) Set the Email parameter to the Email input parameter of the Screen.



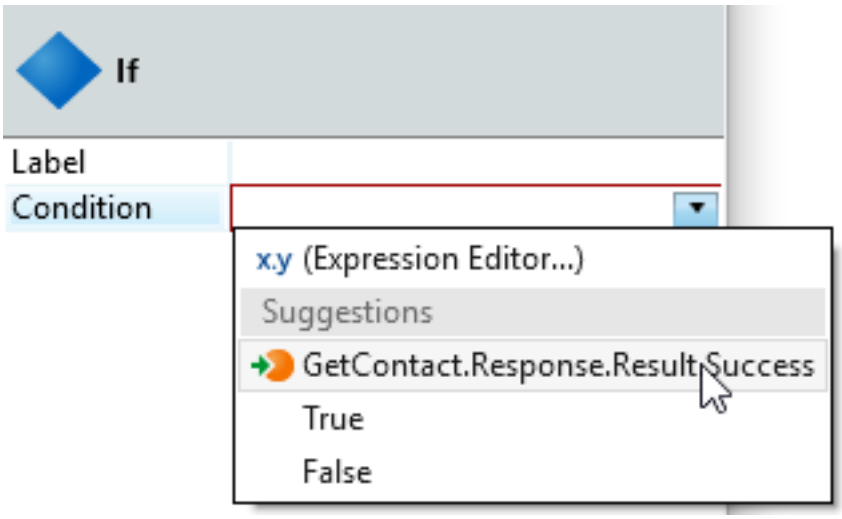
f) Create the **True** branch connector from the If to the *GetContact* invocation created above.

3) Validate if the operation to retrieve the contact information from the external system was successful.

a) Drag another **If** and drop it to the right of the *GetContact*, then create a connector from the *GetContact* to the new *If*.

b) Set the **Condition** of the new *If* to

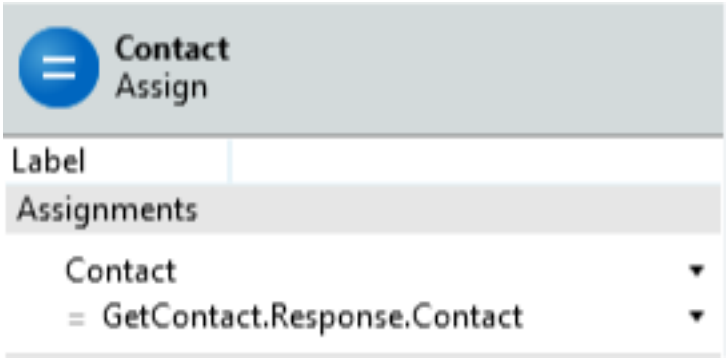
```
GetContact.Response.Result.Success
```



c) Drag an **Assign** and drop it to the right of the *If*, then create the True branch connector between both.

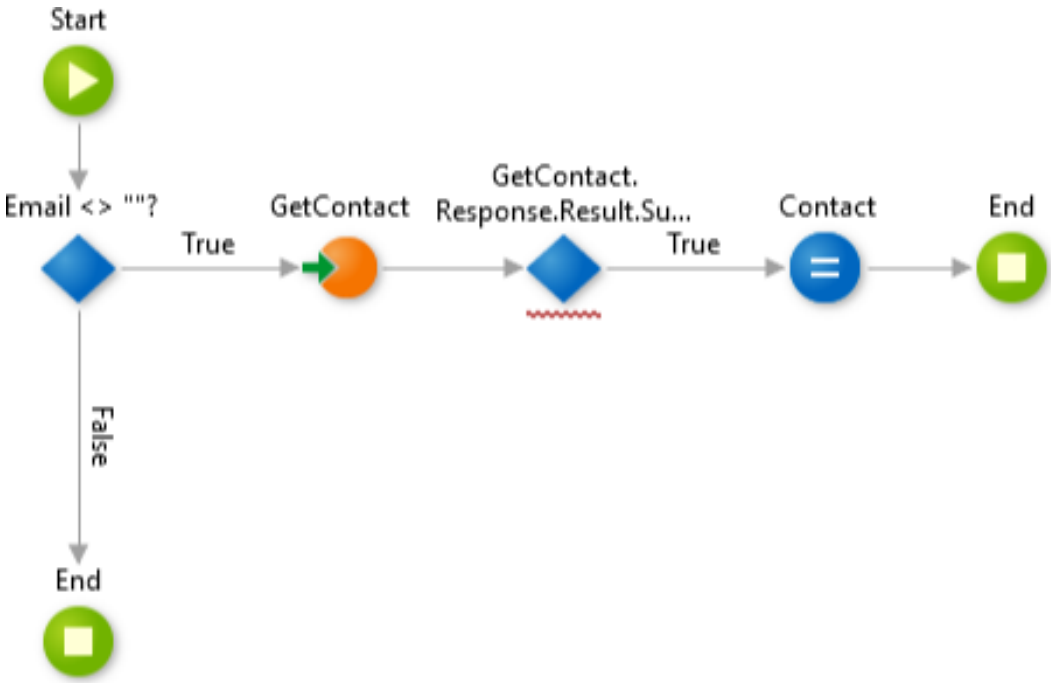
d) Define the following assignment

```
Contact = GetContact.Response.Contact
```



e) Drag an **End** and drop it to the right of the assign and then connect both.

f) The Preparation flow should look like this

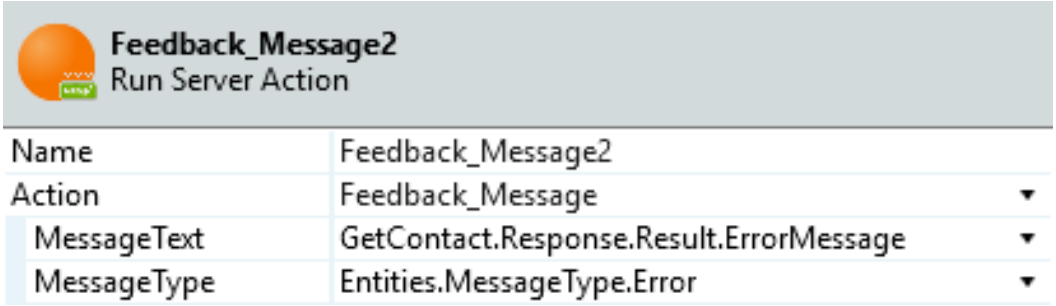


g) Drag a **Run Server Action** and drop it below the If that validates the success of the *GetContact* REST call.

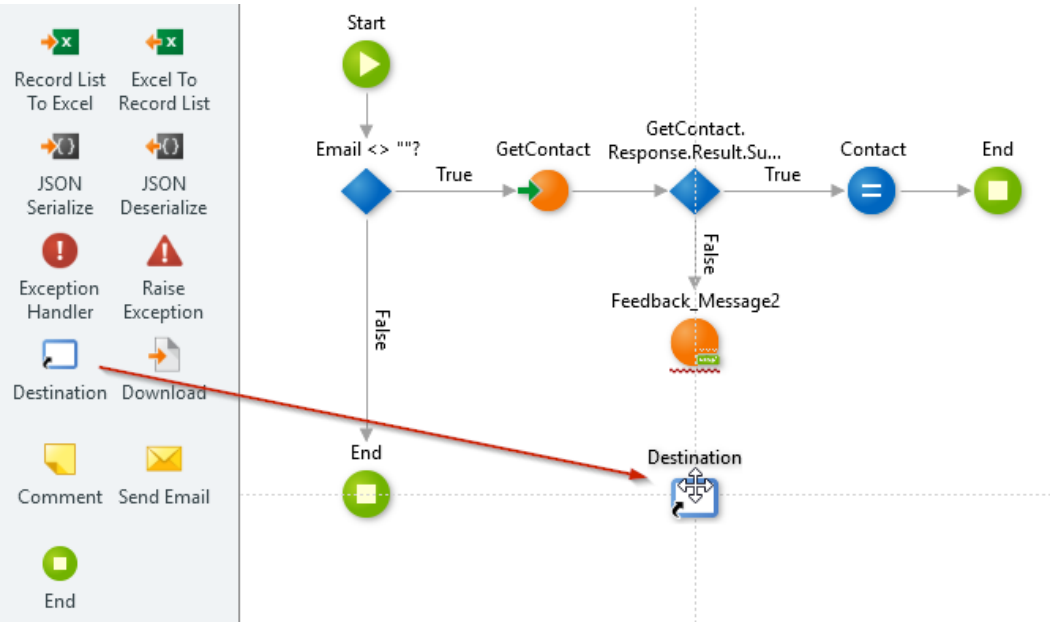
h) In the **Select Action** dialog, choose *Feedback_Message*.

i) Create the *False* branch connector from the If to the action created above.

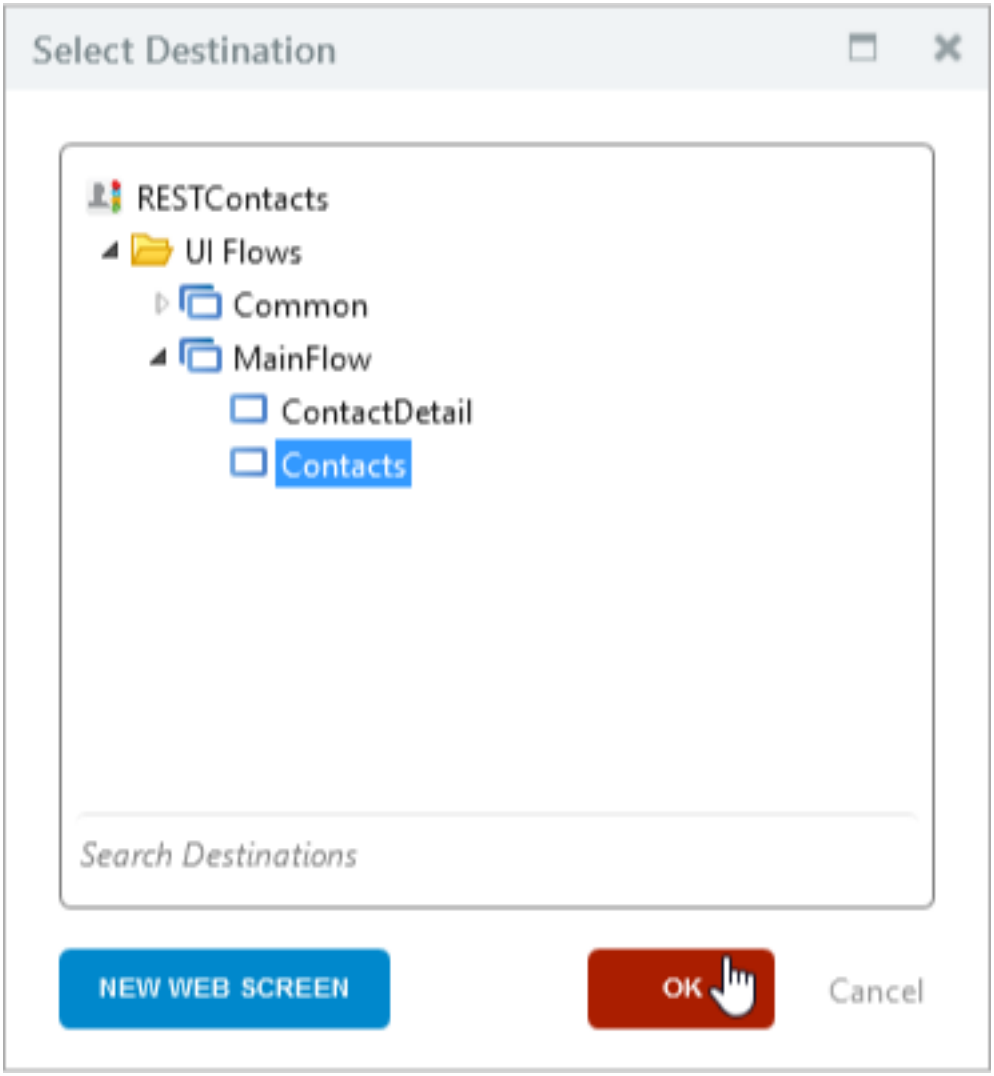
j) Set the parameters of the *Feedback_Message2* as follows



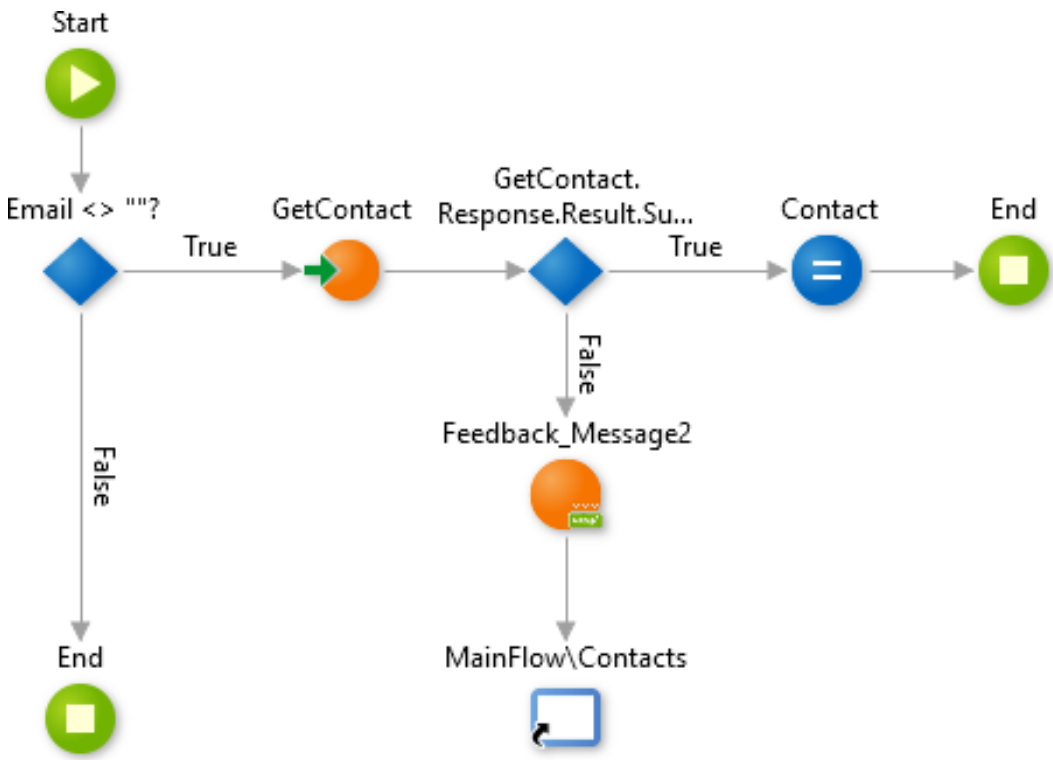
k) Drag a **Destination** and drop it below the Feedback_Message created above



l) In the Select Destination dialog choose the *Contacts* screen



m) Create the connector from the Feedback_Message to the Destination. The flow should now look like this

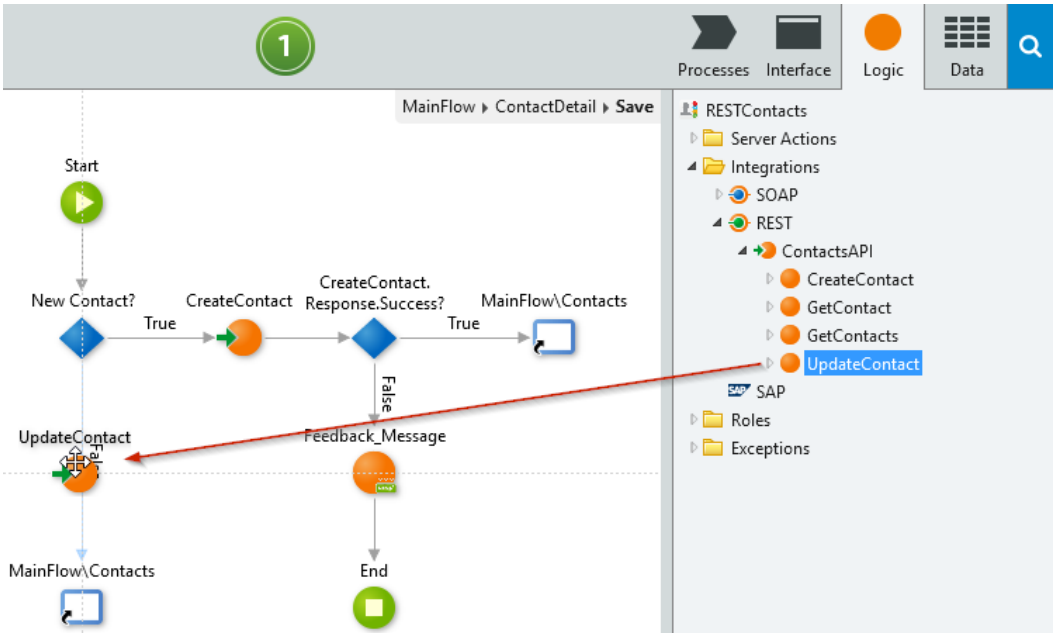


NOTE: The logic defined in the previous steps will ensure that when we navigate from the Contacts screen to the ContactDetail screen (i.e. the Email input parameter has a value), we will retrieve the Contact information from the external system, and (on success) store it in a local variable. This local variable is then used to populate the form input values on the ContactDetail screen. If the operation to retrieve the contact information fails, we show a feedback message, and redirect the user to the Contacts screen.

4) Change the Save Screen Action to invoke the UpdateContact method of the ContactsAPI, to update a contact when the Email input parameter is not empty.

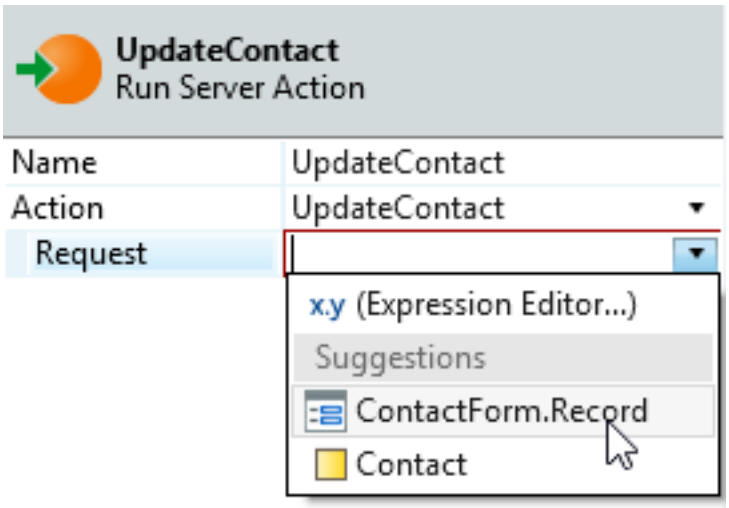
a) Open the **Save** screen action of the ContactDetail screen.

b) From the Logic tab, drag the UpdateContact and drop it on the False branch.

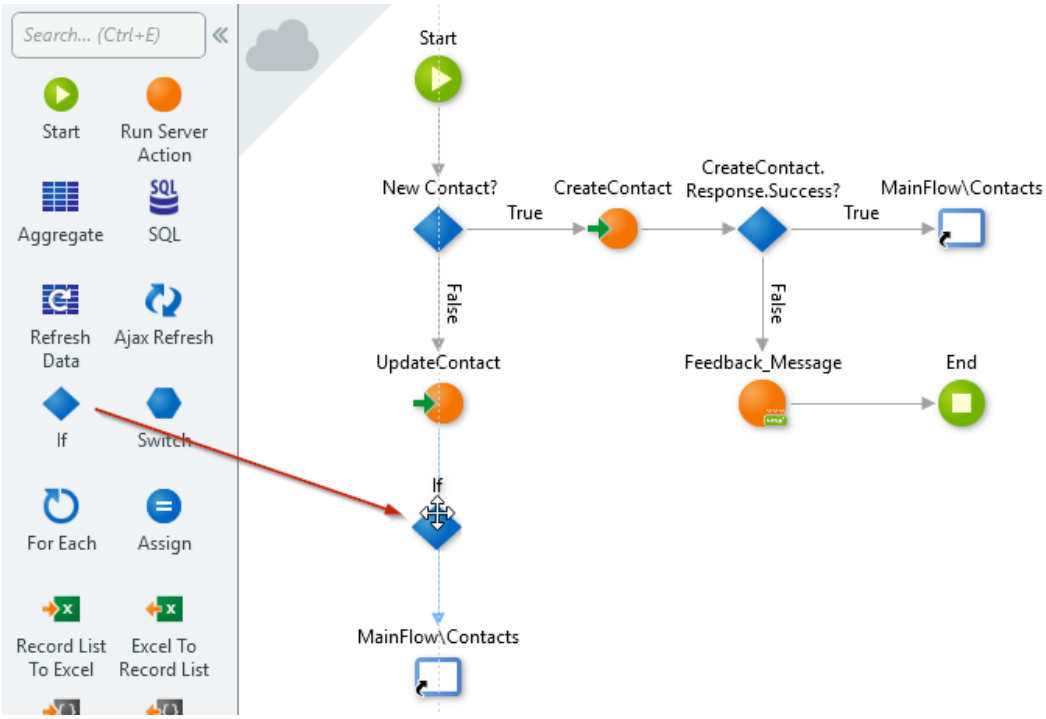


c) Set the Request parameter to

```
ContactForm.Record
```



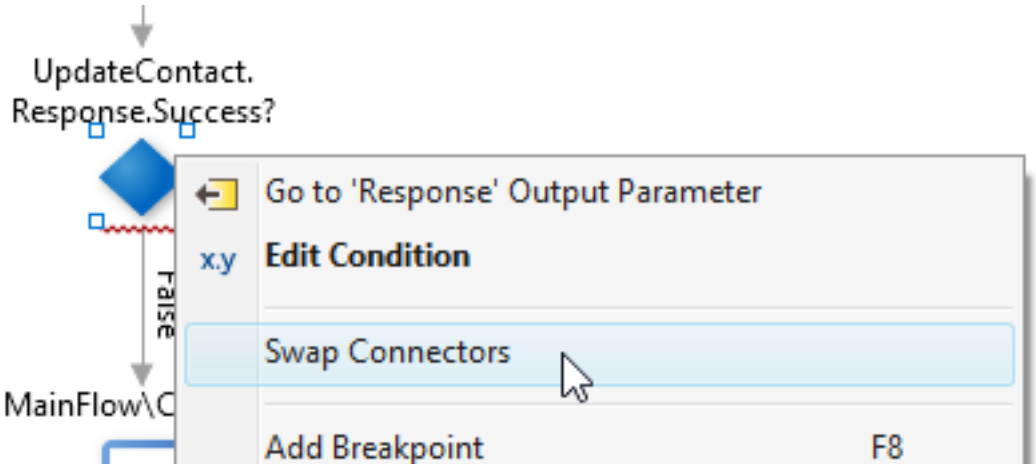
d) Drag an **If** and drop it between the *UpdateContact* and the *MainFlow\Contacts*



e) Set the **Condition** of the If to

```
UpdateContact.Response.Success
```

f) Right-click the new **If** and select *Swap Connectors*



g) Drag a **Run Server Action** and drop it to the right of the If, then in the Select Action dialog choose *Feedback_Message*.

h) Create the False branch connector from the If to the new Feedback_Message (i.e. *Feedback_Message3*).

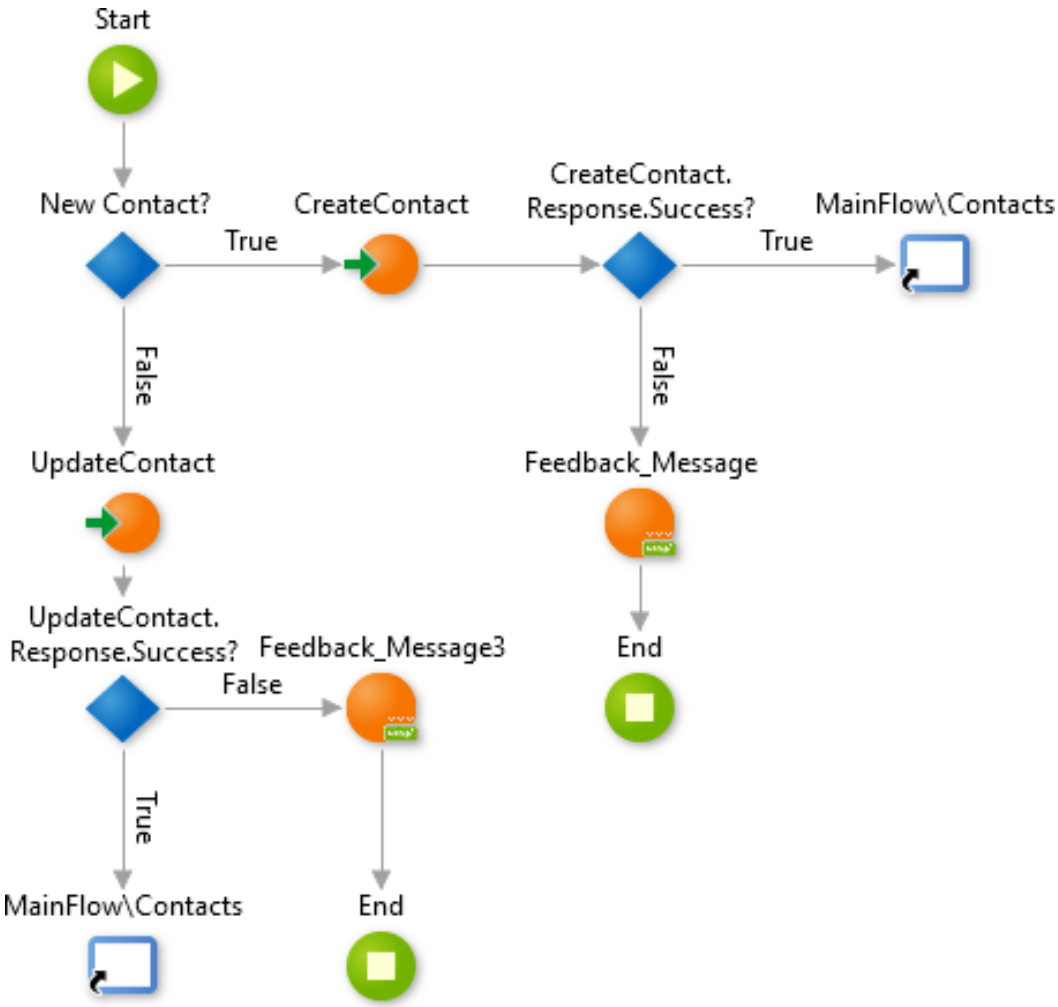
i) Set the parameters of the *Feedback_Message3* as follows

Feedback_Message3
Run Server Action

Name	Feedback_Message3
Action	Feedback_Message
MessageText	UpdateContact.Response.ErrorMessage
MessageType	Entities.MessageType.Error

j) Drag an **End** and drop it below the Feedback_Message created above, then create the connector between both.

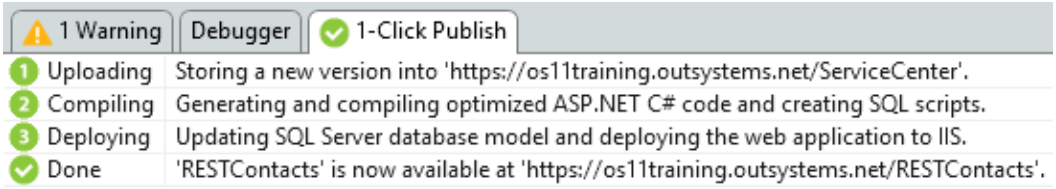
k) The Save action flow should look like this



5) Publish the application using 1-Click Publish button and verify that the publish completed successfully in the 1-Click Publish Tab.

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

   b) Verify in the 1-Click Publish tab that the publishing process was successful.



   c) Preview the app in browser by clicking on the **Open in Browser** button.

   d) Click on the name of one of the contacts listed.

   e) Change some of the data in the form then click Save.

   f) After saving you should be redirect to the Contacts screen, and see the changes on the chosen contact.

End Lab

In this lab, we integrated with a REST Web Service, namely POST and PUT methods to create and update Contacts on the external system.

To accomplish that, we have used the OutSystems visual interface to add a few extra methods that enable the creation of a new contact, and updating a new contact.

Once the new methods were added to the integration, the user interface and underlying logic was changed to allow to create new contacts and changing existing contacts.

At the end of this exercise you should be able to integrate with a simple REST API to write and update data in an OutSystems application.