Chatbots

September 2019

OBJECTIVE

Welcome to the chatbots workshop!

Here we will see the power of the chatbots generator introduced in GeneXus 16.

We'll create a chatbot that answers questions at a technology and business conference.

Chatbots generated with GeneXus run on both Web and Mobile.

SOME PRIOR ADJUSTMENTS

Lab requirements:

GeneXus Version:

GeneXus Beta.

NLP Provider:

In this lab, we'll use Watson as Natural Language Processing (NLP) Provider. We'll skip the configuration of the NLP provider account because it is ready beforehand for the lab.

Knowledge Base:

The knowledge base is located at https://samples.genexusserver.com/v16/versions.aspx?CourseGX29Chatbot,

You have to do a checkout of the **InitialState** version.

Running the workshop app:

You can run it on the web and on a mobile device.

The web object that acts as entry point to the chatbot is the web panel called *RUDICourseGX29ChatbotWebUI*. Running this object in each test cycle is enough.

The entry point SD object is *RUDICourseGX29ChatbotSDUI*.

To run the mobile app, we recommend using your own device. The *Annex: How to run the app on your device* (below in this document), provides configuration instructions.

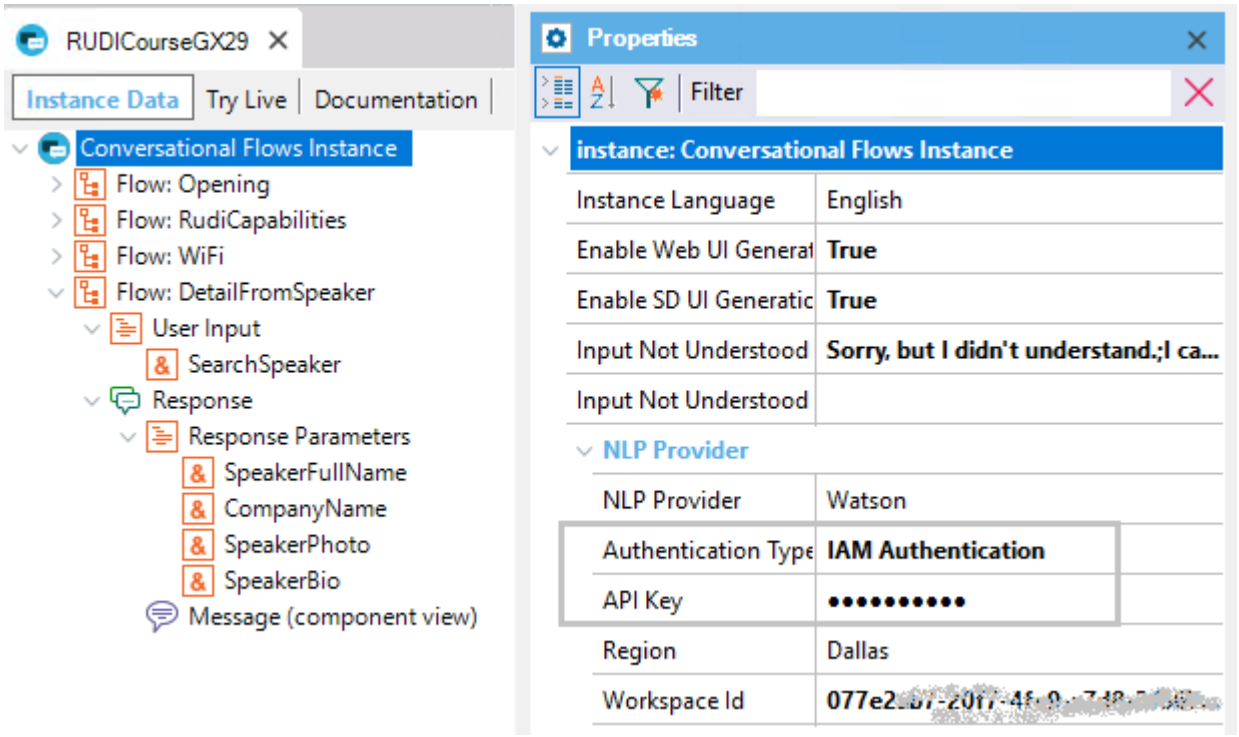
GETTING TO KNOW CHATBOTS IN GENEXUS

First, let's run the following steps to interact with the chatbot.

The chatbot in your KB is called **RUDICourseGX29** and has not yet been synchronized with the Provider.

In the object **RUDICourseGX29** of the parent node "Conversational Flows Instance," set in **API Key** property the value provided with the lab material. The Authentication Type property must be "IAM Authentication," and the Region must be "Dallas."

The workspace ID is automatically assigned.



After saving the changes (the same happens by right-clicking on the object RUDICourseGX29, and selecting the Synchronize Chatbot option), a metadata file is generated that impacts the NLP provider, to generate a dialogue with its intentions and entities.[1]

Given a user's intention, different objects or topics can be distinguished within that intention, which we formally call Entities.

Entities are stored in the NLP provider, with their possible values and synonyms.

To create a chatbot, you need to model its behavior (define intentions, entities) and the response shown to the user for each of those intentions.

NOTE: THE CHATBOTS GENERATOR WORKS AS A PATTERN. FROM THE MODEL, THE PATTERN AUTOMATICALLY GENERATES ALL THE OBJECTS NECESSARY TO IMPLEMENT THE CONVERSATION BETWEEN THE USER AND THE NLP PROVIDER.

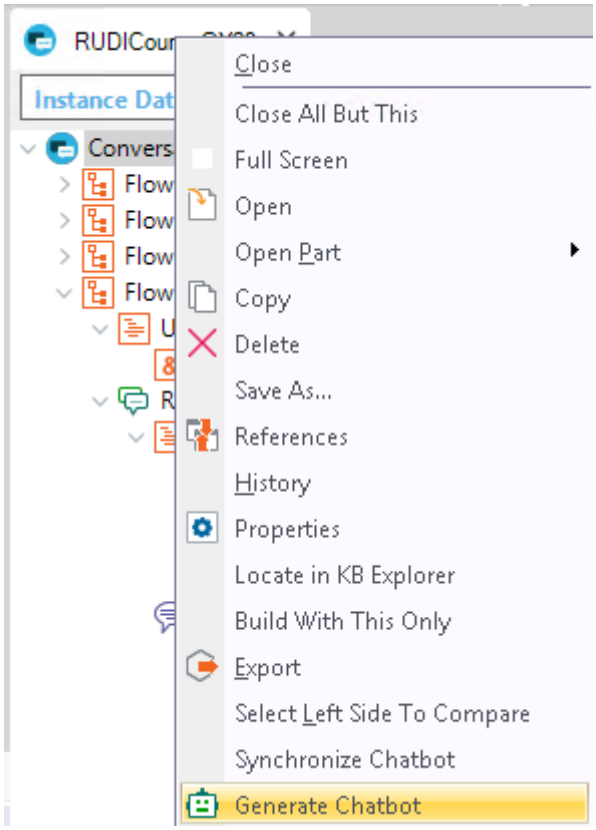
We've already understood the main concepts, and described the Chatbot Generator. So, let's start building!

THE CHATBOT IN ACTION

Through our chatbot, users can:

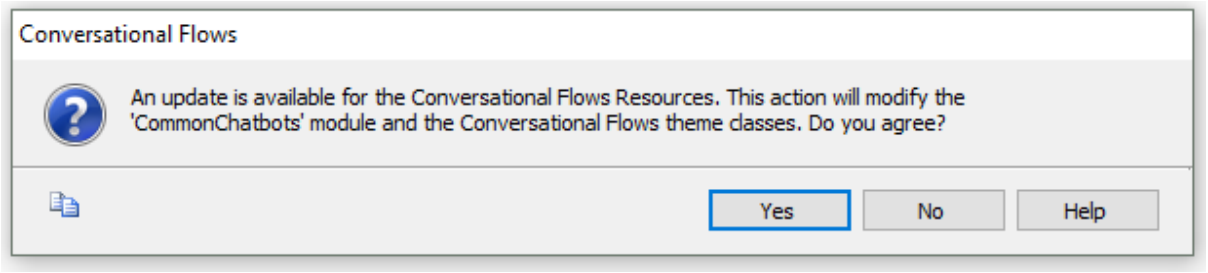
- Access Frequently Asked Questions (for example: WIFI password)
- Know where some services are located
- Obtain speakers' information

Right-click on the object **RUDICourseGX29** (Conversational Flows object) and then on Generate ChatBot.



Note: The "Generate Chatbot" option makes the objects that will be generated from our instance (RUDILabGX29) to be calculated and imported into the KB.

The following message may appear, indicating that the chatbots generator resources will be updated (if not already up-to-date).



Run a **Rebuild all**.

The Create DB action will be requested. Note that there are two tables called GXChatMessage and GXChatUser, which are used by the chatbot.

INITIALIZATION

The entities have been created in the NLP provider, but their values must be initialized.

We'll be working with the following entities:

- Floors
- Rooms
- Speakers

The values will be loaded in the NLP provider through an API provided by the chatbots module.

Open the **BatchEntities** object to get to know the method used to upload values and entities to Watson ("Chatbot.SendEntityValues").

Run the BatchEntities procedure (right-click and select Run from the tab).
This process trains the system, so although the request to the NLP provider immediately returns an HTTP Status 200, it may take a few minutes to complete the training process.

The following will be displayed in the GeneXus output:

Floors: [{"Id":"","Type":2,"Description":"200 "}]
Rooms: [{"Id":"","Type":2,"Description":"200 "}]
Speakers: [{"Id":"","Type":2,"Description":"200 "}]

STARTING TO BUILD THE CHATBOT

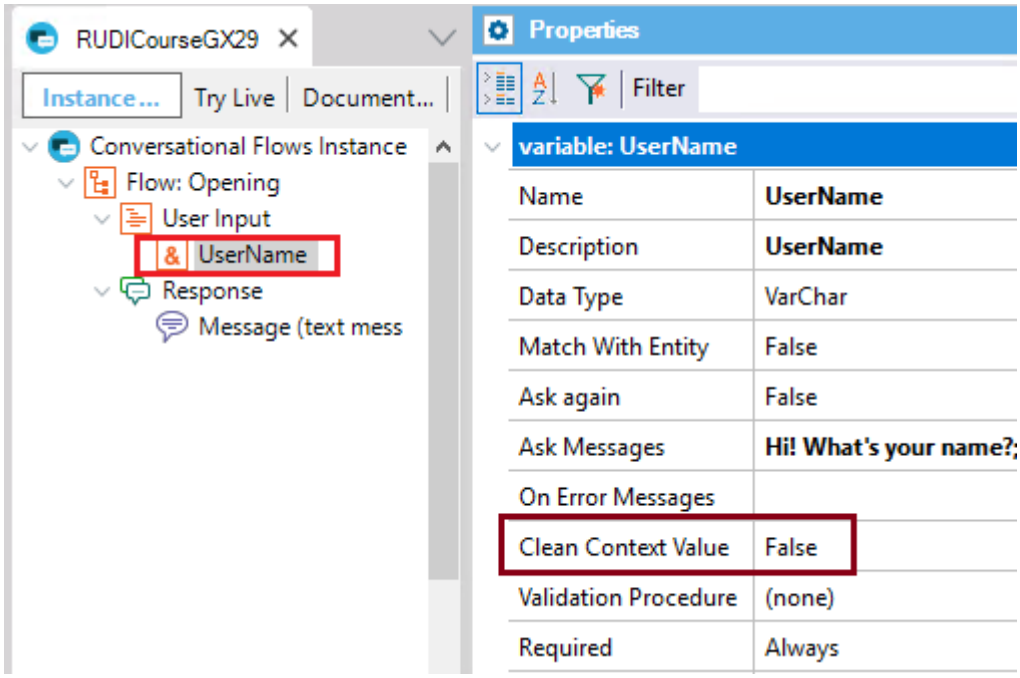
Note some interesting flows:

1. GREETING

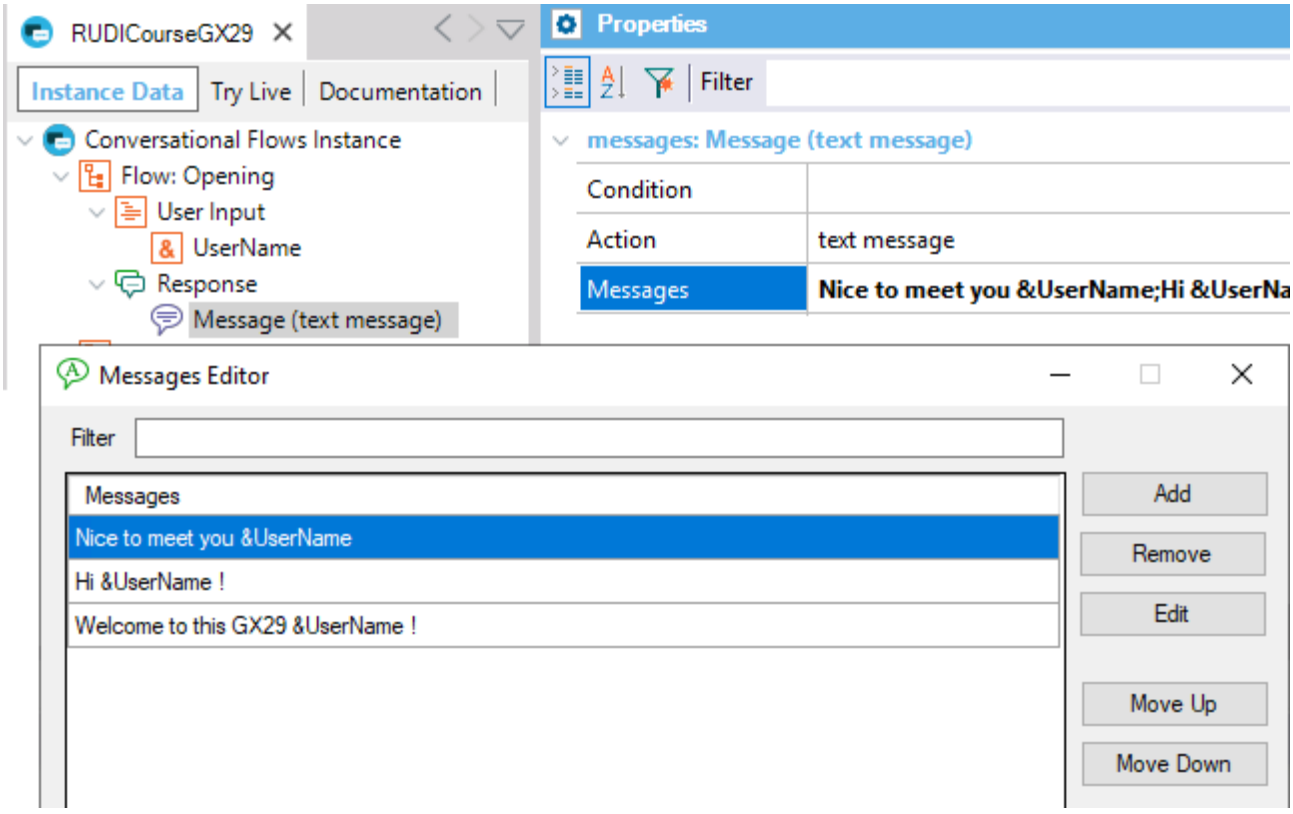
First of all, let's look at the simplest flow, which is the greeting ("Opening" flow). This flow may not have any User Input (but in this case we want to ask the user's name here, so that the chatbot is more friendly), and return the greeting.

Note the **User Input** "UserName" that has the property **Clean Context Value** = FALSE. This means that the user's name will be saved in the context [(2)] (an essential tool in a chatbot so that the user doesn't have to repeat information he/she has

already provided).

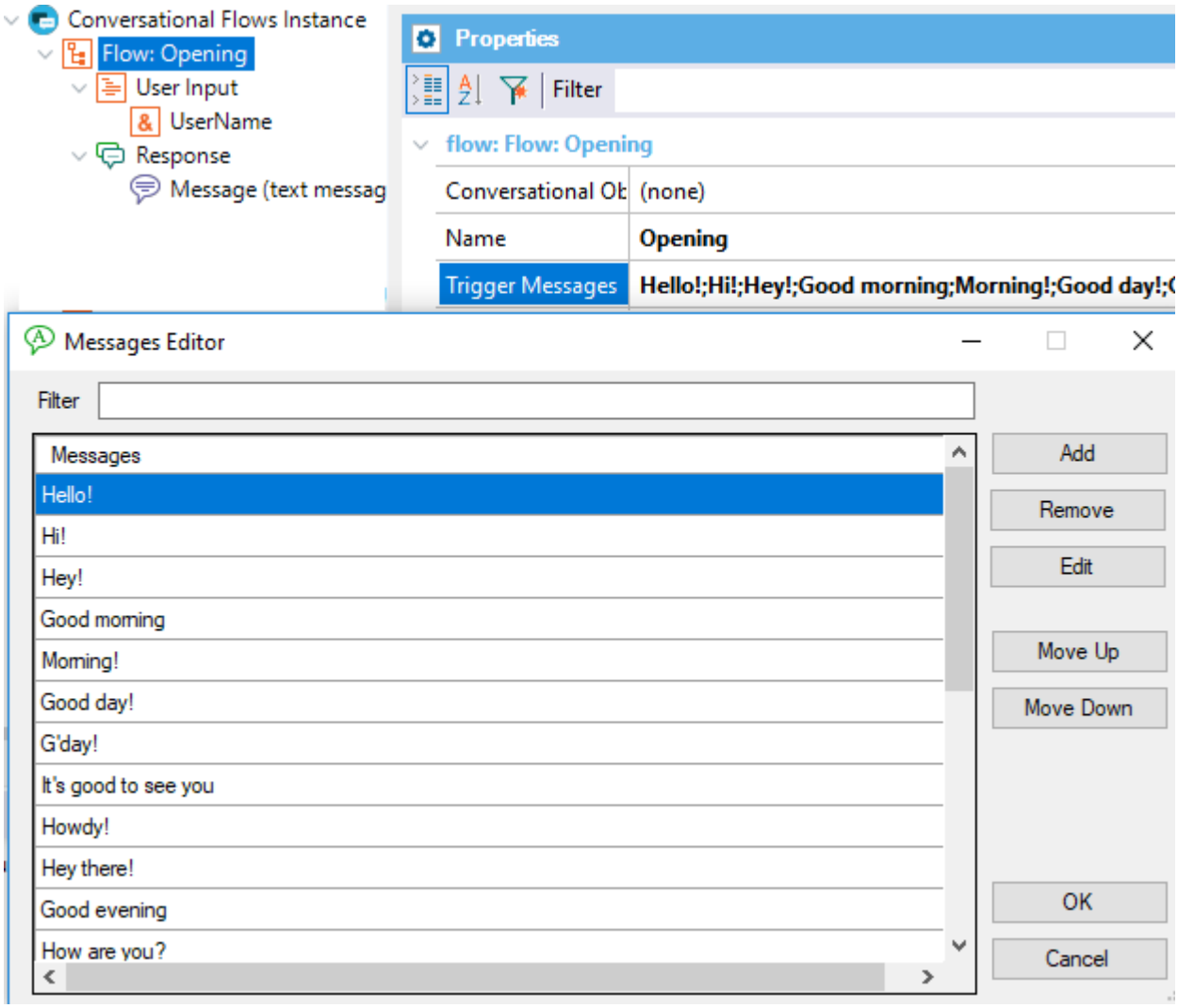


Note the answer as well. It's a Message node of text type where an answer is provided to the user, instantiating his name.

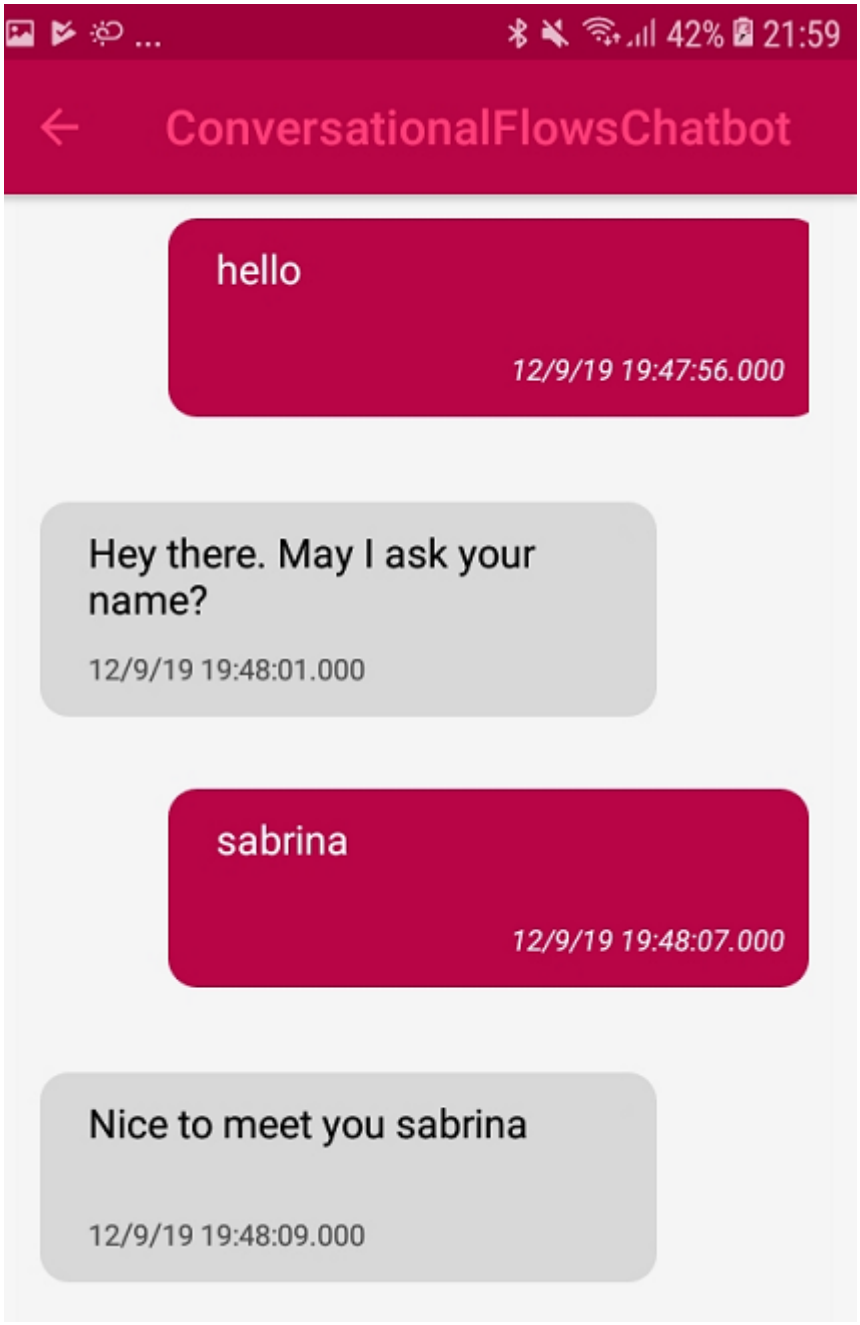


How does the NLP Provider identifies the user's intention, which in this case is a greeting?

It does it based on the training given to the flow, using sample messages (called "**Trigger Messages**" in GeneXus).
You can see them by clicking on the **"Opening"** node of the flow, in the property Trigger Messages.

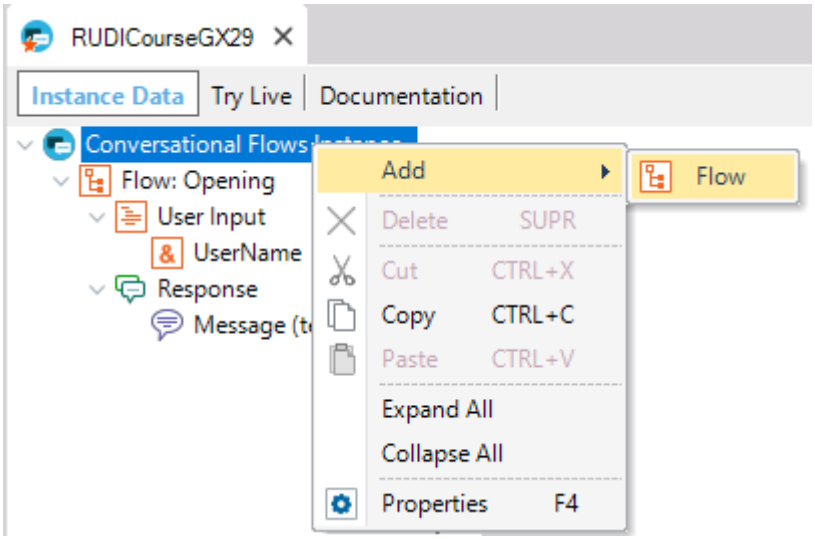


At runtime, you'll see something like this:

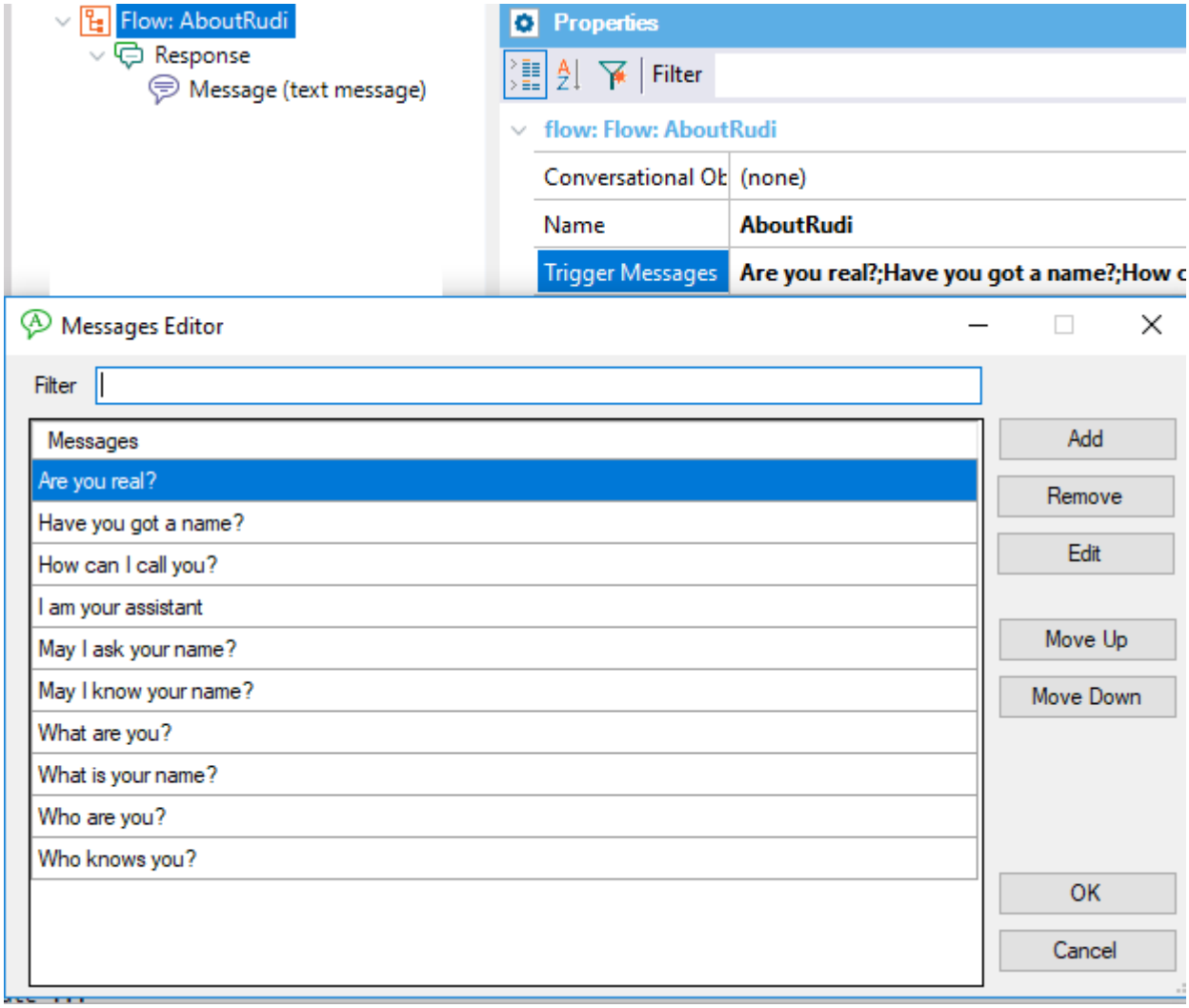


Now build a similar flow where the user can find out who RUDI is.
To do so, follow the steps below:

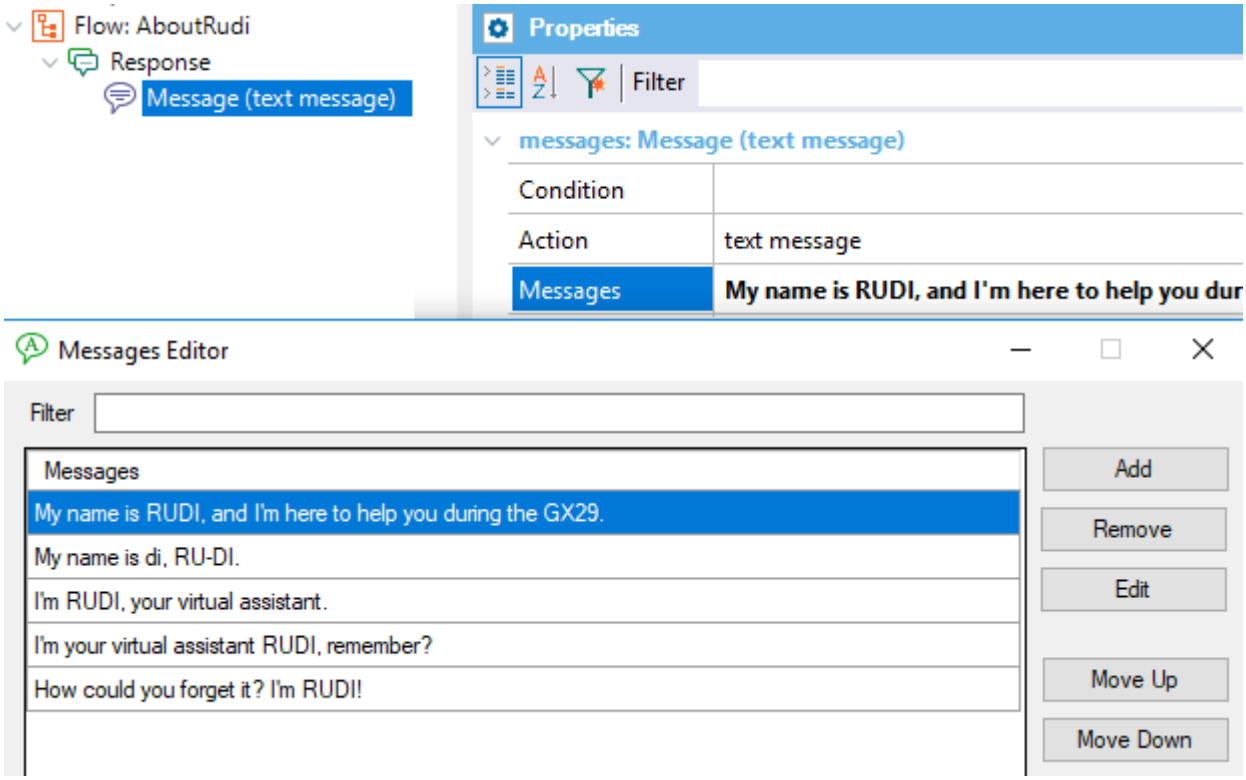
- Create a new Flow called “AboutRudi.” Right-click on the node **Conversational Flows Instance** and then on Add - Flow.



- Add Trigger Messages (those you consider necessary; in a real situation, the more complete and high-quality the training, the better the identification of the user's intention will be).

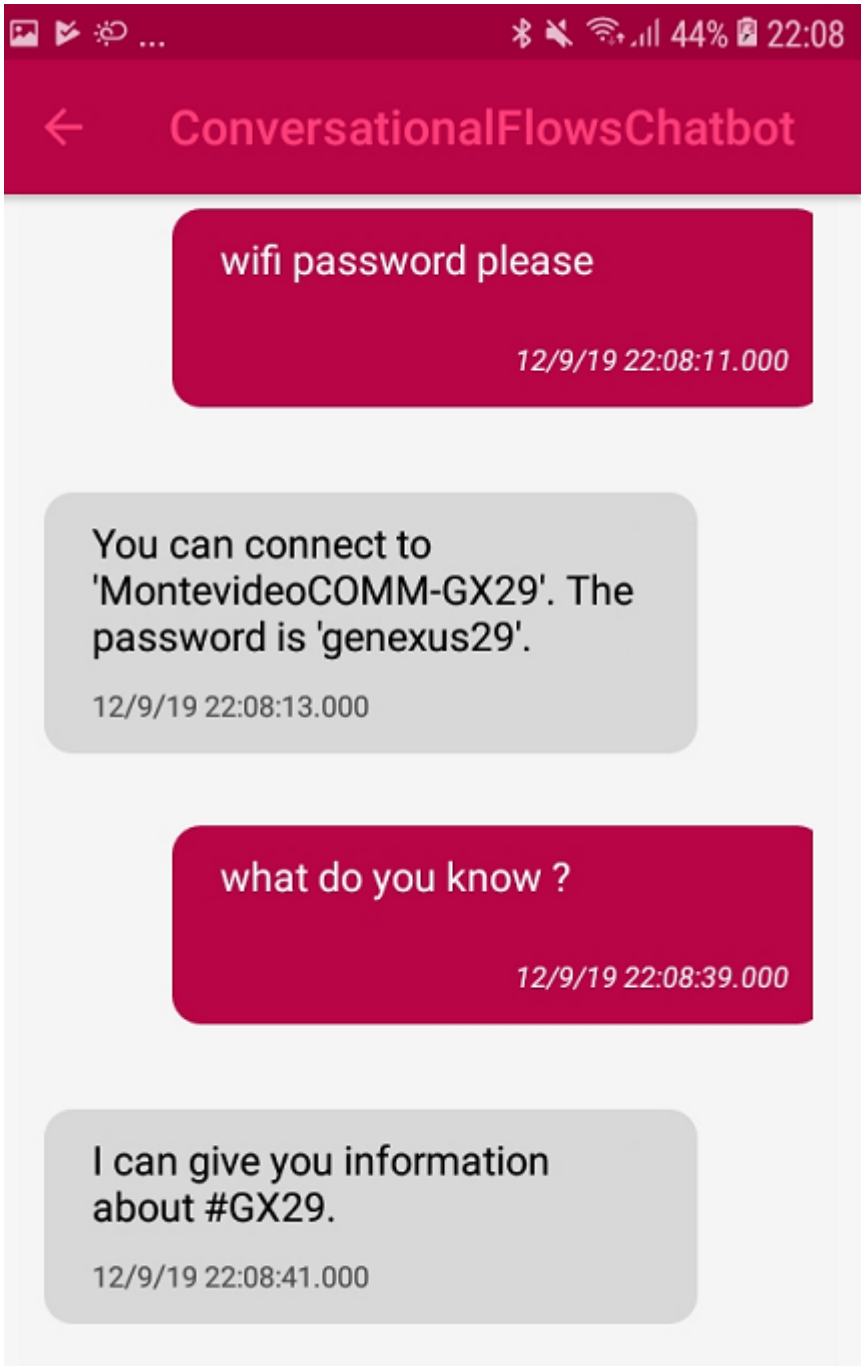


- To add replies, right-click on the Response node, and add a Message. There you can add all the answers you want. The answer given to the user will be random, and it is precisely what makes the UX more “natural.”



There are several flows of the same style that will be included in the chatbot, for example, knowing the WIFI password, knowing what RUDI talks about, etc.

Let's see then how our chatbot responds!



2. ACCESS SERVICES

We will make a flow where the user will be able to ask about a service, for example, the restroom. To give him the answer, he will be asked the floor where he is located.

- Create a flow called “ServiceLocationBathRoomFloor.”

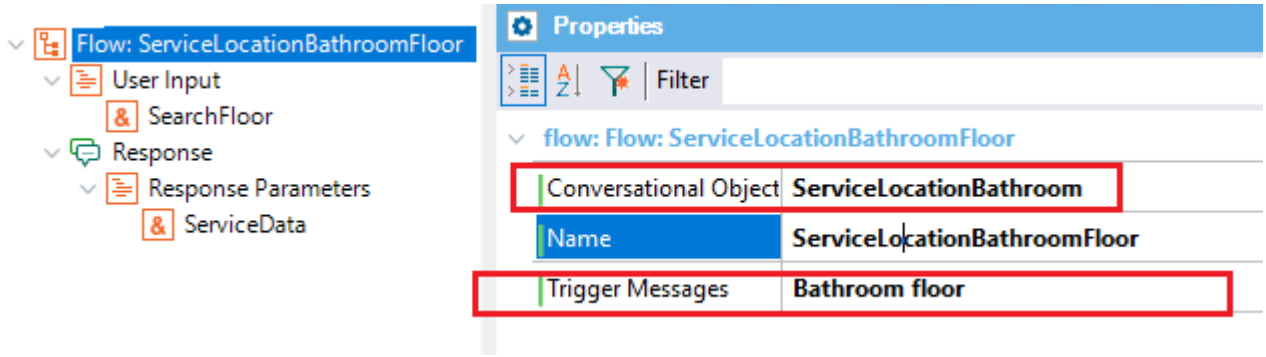
- In the Trigger Messages property, configure an example (or as many as you want) for triggering this intention; for instance: “bathroom floor.”

  How is the answer implemented?

  When the user finishes entering the requested data (in this case, the floor), the given object is executed in the “Conversational Object” property.

- In the “Conversational Object” property, configure the “ServiceLocationBathRoom” procedure (it’s a proc stored in the KB); given the floor where the user is located, it returns the nearest restroom.

  Note that when you set up this property, a **User Input** (“SearchFloor”) and a **Response Parameter** (“ServiceData”) are automatically assigned. These match the input and output parameters of the proc “ServiceLocationBathRoom,” respectively.



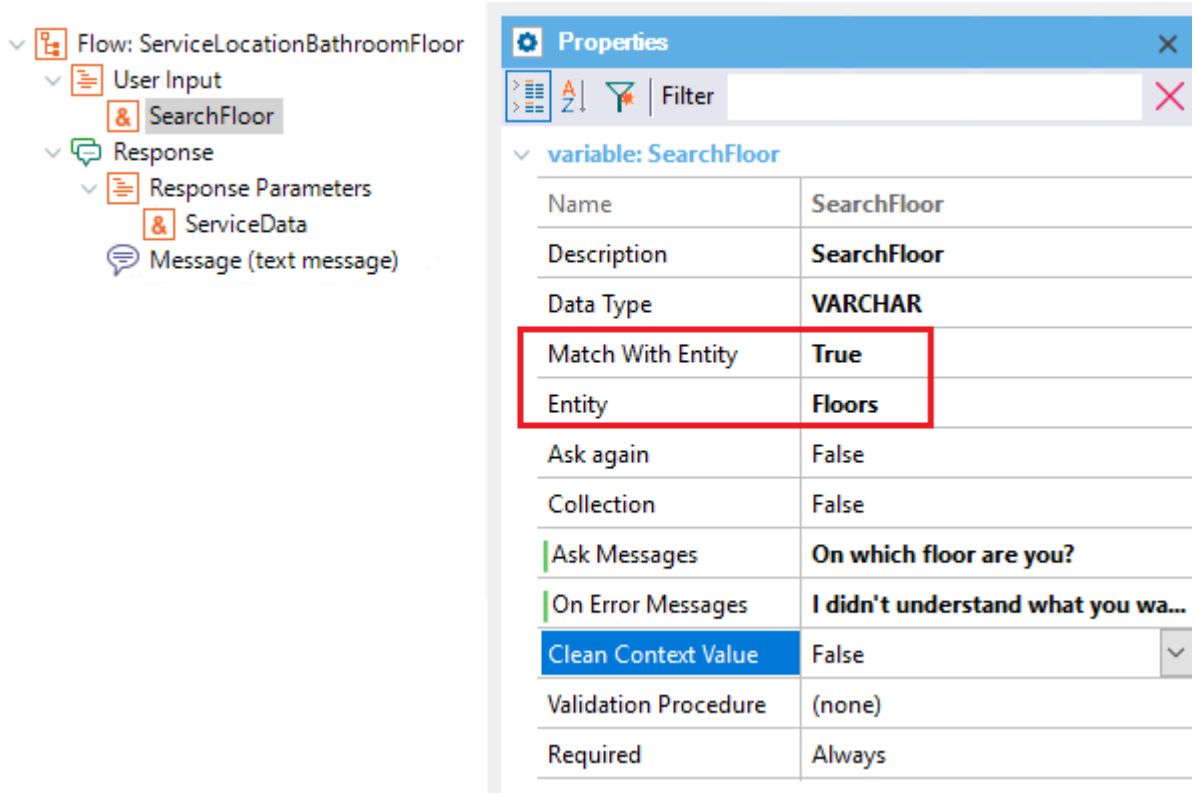
The User Input “SearchFloor” is where the user indicates the floor where he is in. This User Input has **Match with Entity = TRUE**, and the entity is **Floors**.

What does Match with Entity mean?

That a check will be made so that the user doesn’t enter data that is not within the values (and their synonyms) of this entity defined in the NLP Provider. It's like an “integrity check” against the Provider.

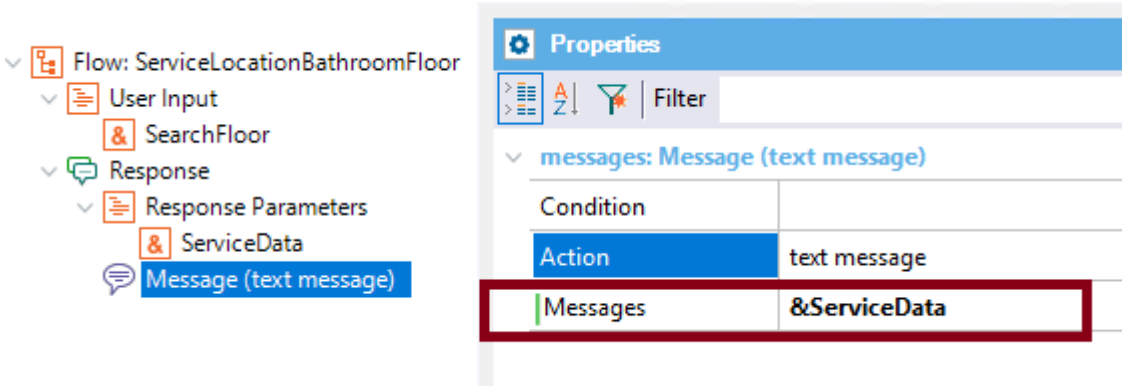
You can open the “EntityFloors” object where we load the values of the “Floors” entity in the Provider (floors and possible synonyms) to clarify what data is in the NLP Provider.

That is, if the user enters a floor that is not within the values of the entity, the Provider shows the error, and gives feedback to the user with the message indicated in the property **On Error Messages** (E.g. “I didn't understand what you meant by '&GXUserInput', but there are restrooms on every floor.)

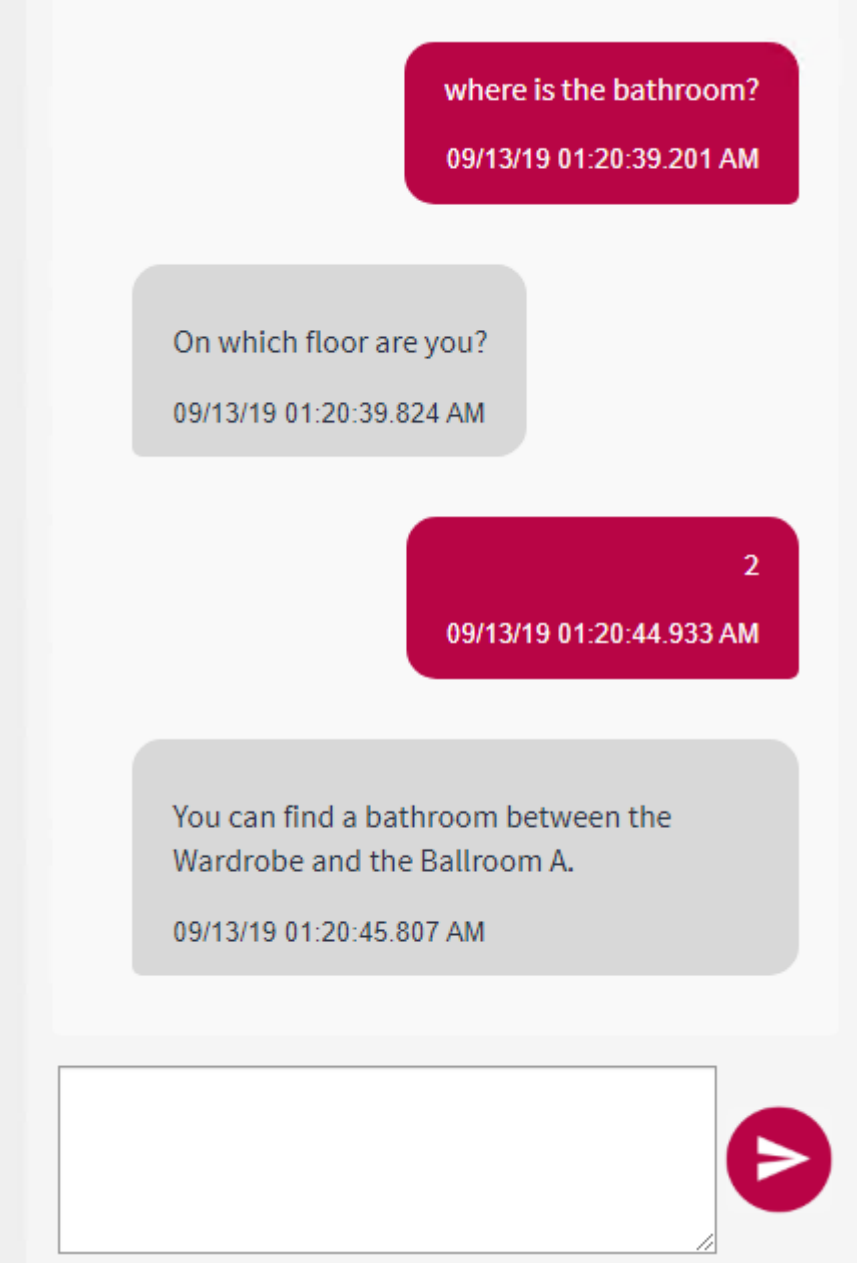


- The only thing left is to add a Message so that it responds to the user when he indicates a valid floor. What will be answered is the output variable of our Conversational Object (&ServiceData).

Right-click on the "**Response**" node, click on "**Add message**," and add &ServiceData in the **Messages** property of the Message node.



That's all; you can test it at runtime:

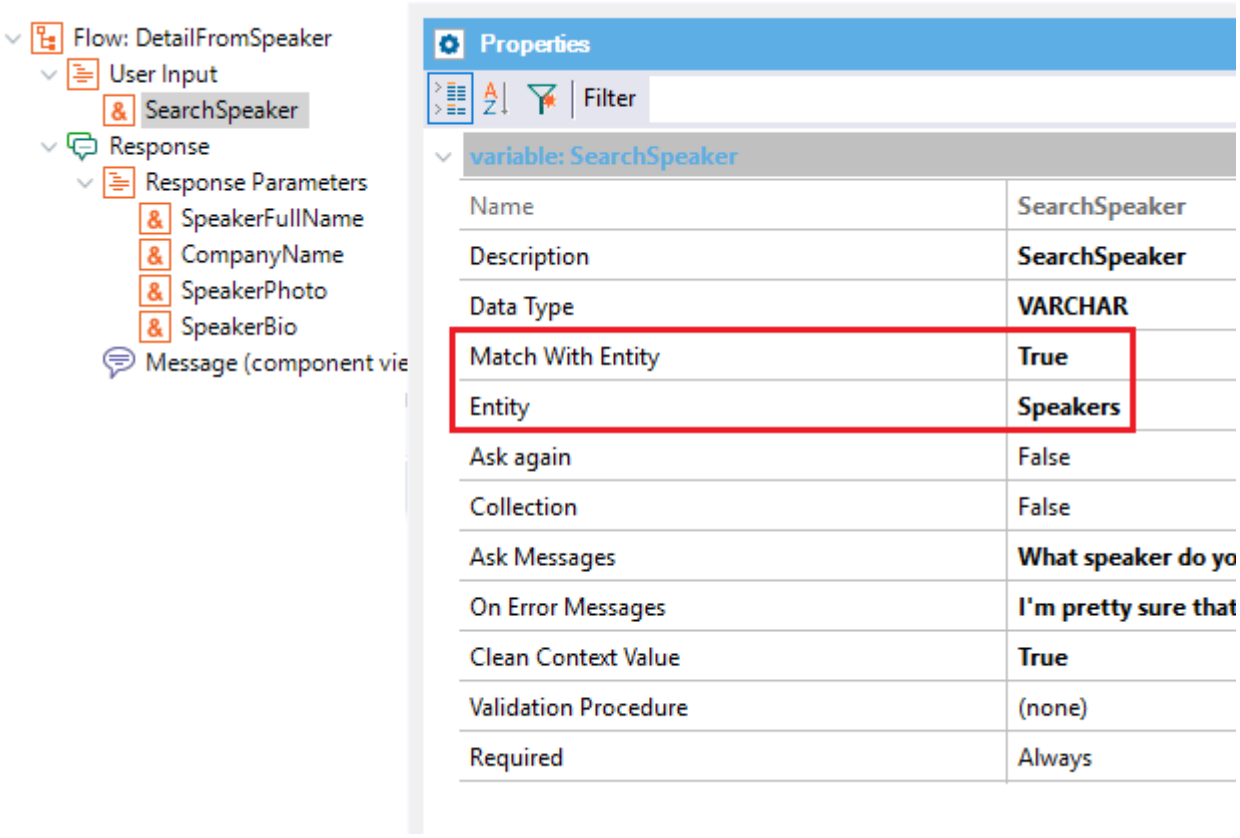


3. SEE A SPEAKER'S DETAILS.

This flow ("DetailFromSpeaker") is intended to graphically show (without using text) the speaker's details. Therefore, it has a **User Input** that is the speaker's name. If the speaker is mentioned in the user's initial message, it is not requested.

Note that the speaker has the property **Match with Entity** = TRUE, and

Entity=**Speakers**.



In this example, the speaker's information will be displayed in a panel (Web or Mobile), as appropriate.

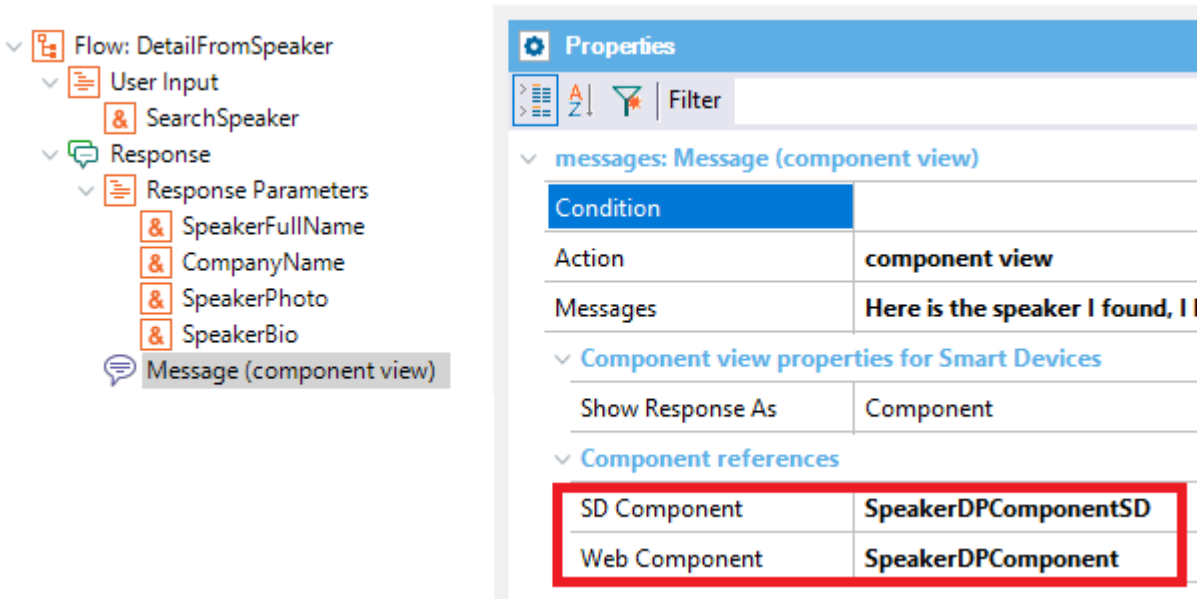
The speaker's information is obtained using a Data Provider, which is invoked from the panels that will be used to show the result of the user's query.

The web object that shows the result is **SpeakerDPComponent**, and for SD it is **SpeakerDPComponentSD**. You can open them and see the call to the Data Provider which given a speaker's name, return his details.

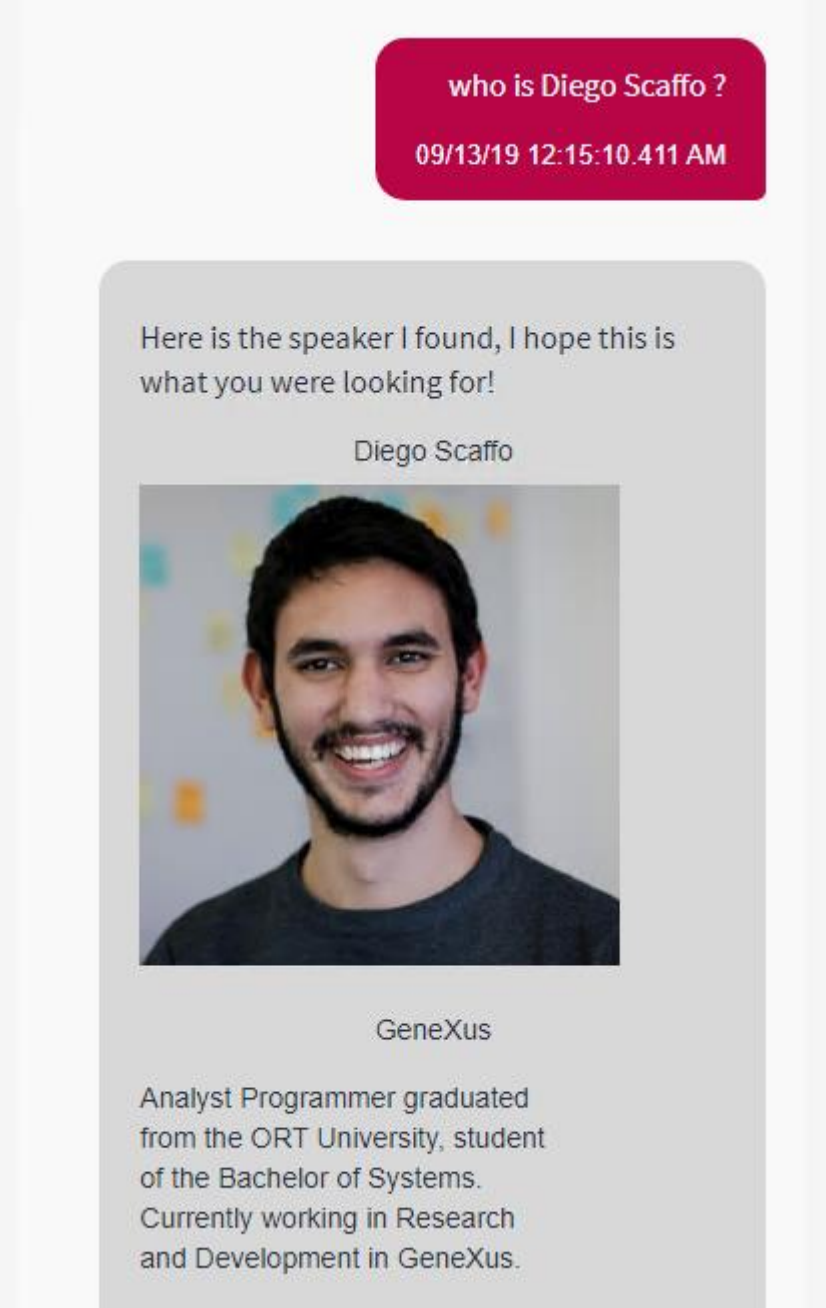
Let's see the flow configuration:

The panels mentioned before, where the speaker's details will be displayed are configured in the Message node, which in this case has the **Action** property set to Component view. The SD and Web panels are configured in the

SD Component and **Web Component** properties, respectively.



Let’s try it at runtime on the web!



And on a mobile:

We've seen some concepts about how to build a chatbot, but there is a lot more! You can read the documentation provided below.

Thank you for participating!

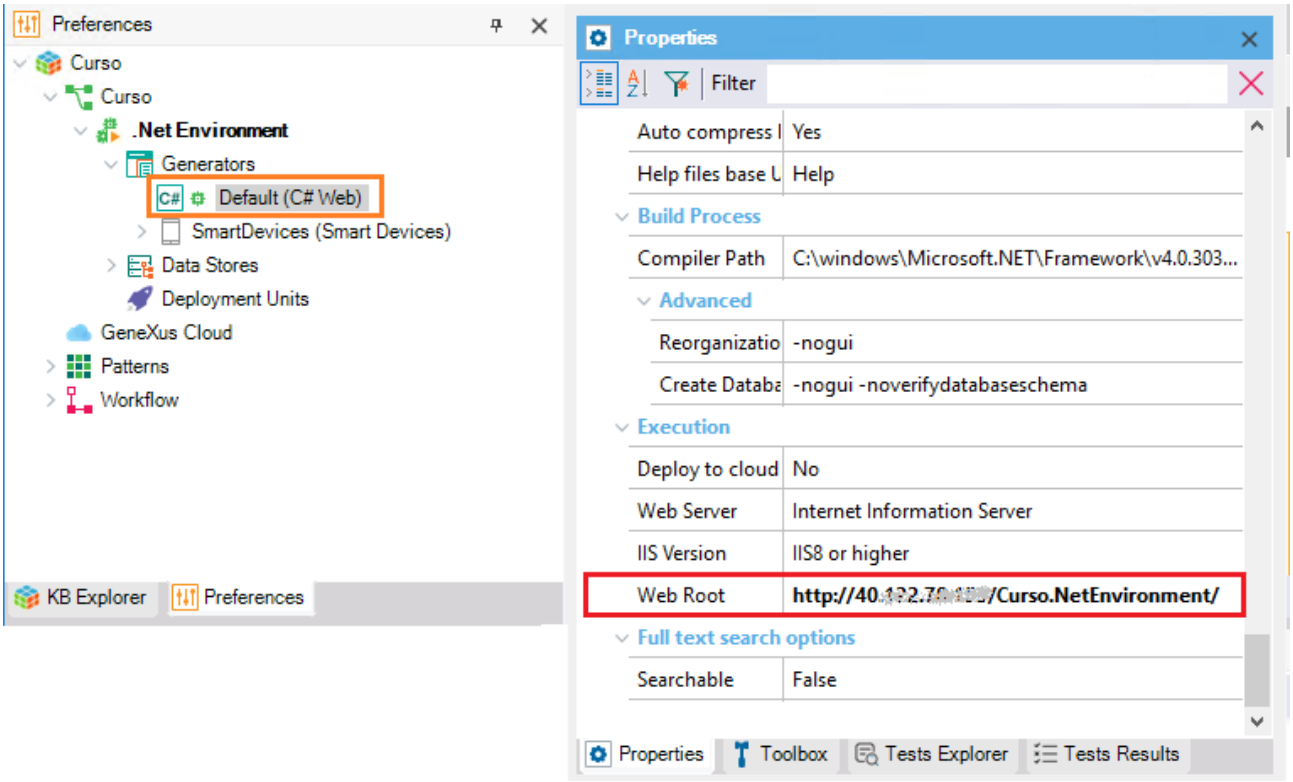
ANNEX: HOW TO RUN THE APP ON YOUR DEVICE

In this workshop you can use an Android or iOS device.

You must obtain the public IP of the machine (click here) and set the properties:

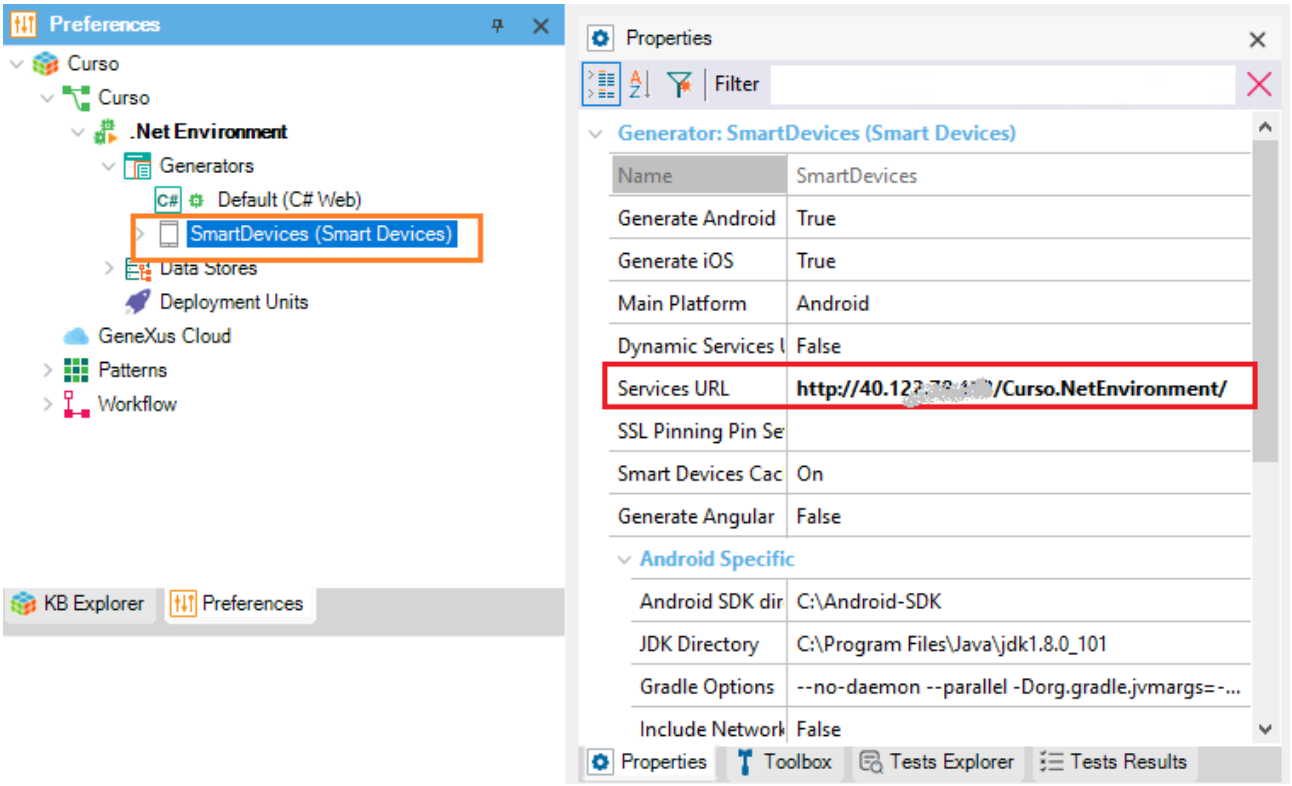
- Web Root (of the C# Generator)

  Change “localhost” for the public IP of your machine.



- URL Services (Smart Devices)

  Change “localhost” for the public IP of your machine.

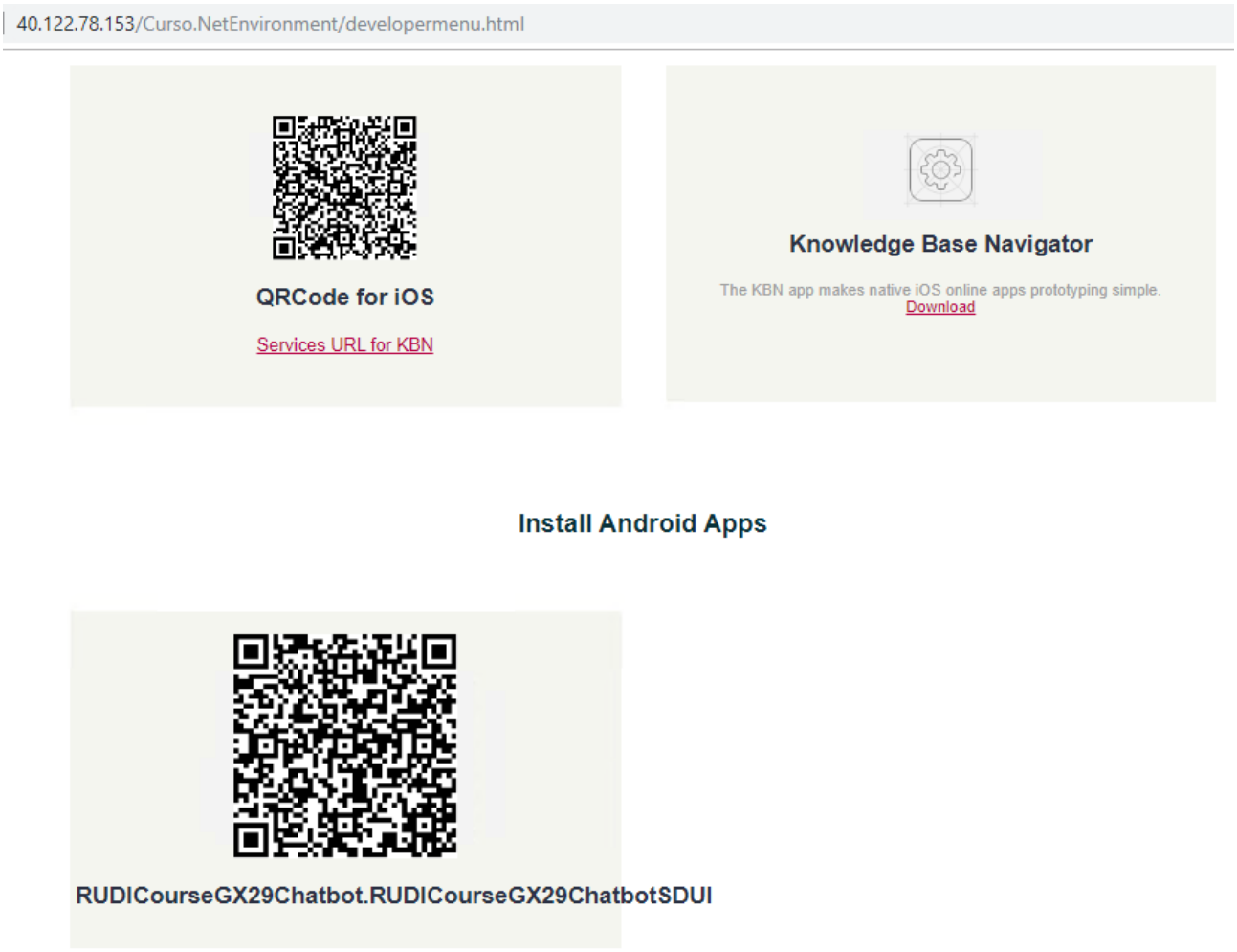


The object **RUDICourseGX29ChatbotSDUI** (the main SD panel) is set as Startup Object, so you can scan its QR Code to install the app on your Android device.

For iOS, you will need to have previously installed the app “GeneXus 16 KB Navigator” (KBN) which can be downloaded from the Apple Store.

Then you can install the app by scanning the QR code,

QR Codes are available through the GeneXus menu option **View -> Show QR Codes**. You should see a screen similar to the one below:



Documentation

[Chatbots in GeneXus]
[Chatbot Generator]
[Chatbots Architecture]
[How to build a Chatbot using GeneXus]
[Chatbots Channels API]
[Chatbot Generator Try Live]

References

[1] For example, given the **intention** to "Obtain a speaker's details," an **entity** of that intention would be "Speakers," and the **value of that entity**, for example, could be "Gaston Milano."

So, a group of several values is known as an entity. One or more of them will be instantiated in the query.

(2) Context is essential in a chatbot. It is the status of the conversation, and is maintained so as not to have to ask the user for information more than once. In a GeneXus chatbot, all variables that are User Inputs are part of the context. Using the property **Clean Context value**, you can clear the value of the context variable. The context can be managed through the Context API.