Genesys Intelligent Automation Bots Integration Guide

Integrating Intelligent Automation with Google Dialogflow

4/13/2025

Contents

Integrating Intelligent Automation with Google Dialogflow

Configuring IA with Dialogflow

To integrate with Google Dialogflow, complete the following steps:

1. Configure NLU Settings
2. Map Intents to modules
3. Map Slots to questions

> ### Important
>
> - Before configuring the bots, please ensure that the server time and the local server time are synchronised.
> - Proxy servers to access *.googleapis.com are not supported.
> - Only HTTPS protocols are supported. For Natural Language Modules, Java clients are supported. For Custom Natural Language Module, REST clients are supported.
> - By default, the 443 port is used by Google Services.

Configure NLU Settings

Open a **Natural Language Menu** module and click the **NLU Settings** tab.

From the **NLU Engine** menu, select Google Dialogflow.

- Leave the **Use default credentials** option checked to use these credentials or clear the option to override them for this particular application.

Map Intents to modules

Intelligent Automation reads all Intents, Utterances, and Slots associated with a domain and then displays that information on the **Intents List** page for the **Natural Language Menu** module. This is where you'll map each Intent to a module.

To map an Intent to a module, simply select a module from the **Intelligent Automation Module to**

Trigger menu for the Intent. Intelligent Automation will then automatically update the application's callflow accordingly.

> ## Important
>
> If the fulfillment is being performed by the bot, you can select *None* from the **Intelligent Automation Module to Trigger** menu and a generic “success” result will be returned after the intent has been completed.

Map Slots to questions

In the natural language engine, an Intent contains Slots, which are key pieces of information you need to extract from the end user to process a request. For example, if a user wants the weather forecast, you would need to know two key pieces of information before providing a weather forecast - city and date. These would be considered Slots.

For each Intent, you should have an associated module containing questions that will extract the right information from the customer. For the Weather example, you would have a Weather Questions module, which contains two questions: *For what city* and *For what date*? Both of these questions relate to the weatherLocation and weatherDate Slots.

Once you have created this module in Intelligent Automation, you need to map it to its associated Slots, as follows:

1. Open the module that is linked to Intent.
2. For each Question block in that module for which you would expect to have a slot, open the block and go to **Question Options**.
3. Check the **Store Answer as a Variable** checkbox and enter the Slot name (weatherLocation).

When a chat or voice session reaches one of those Question blocks, it first checks if that slot has been sent to Intelligent Automation from the natural language engine. In that case, Intelligent Automation uses that as the answer to the question. Otherwise, Intelligent Automation asks the question and waits for a response.

Supported NLU Features

Currently, Intelligent Automation supports the following Dialogflow features:

- Support for Live Agent Hand-off
- Support for Maximum Retries
- Pass Context Settings
- Bypass Opening Question
- Support Follow-up Intents
- Support Fallback Intents
- Support Phrase Hints
- Support for System entities

Support for Live Agent Hand-off

When the live agent hand-off flag is returned as part of the DialogFlow intent, you can configure IA to follow a special path, usually a hand-off to a live agent.

When a value is provided in the **Path to take When Agent Requested** setting, the call is transferred to a live agent defined in the callflow.

In Dialogflow, the liveAgentHandoff flag should be enabled using a Custom Payload response within the intent.

```
{ “liveAgentHandoff” : true }
```

Support for Maximum Attempts

You can set the maximum number of retries to fill a slot and the maximum number of attempts to identify intents in a chat session.

The **Enable Maximum Attempts Counters** setting allows configuring the number of retry attempts and the action to be performed when the limit is reached.

The **Enforce Maximum Attempts for filling Mandatory Slot** will enable or disable the slot fill failure timeout feature. The number of unsuccessful tries can be configured after which the call is either transferred to a different module or return a result.

For example, if the number of retries is specified as 2, the bot will look if it can fill up any slots based on the user input two times. If the slot is not filled after both attempts, the call flow will be transferred to either a different module or return a special value.

When a value is provided in the **Maximum Attempts to Fill Mandatory Slot Values** option, the bot will try to slot-fill the user input. If the bot cannot perform a slot-fill successfully, the counter is increased. When the counter value exceeds the maximum retry values, Intelligent Automation either transfers the callflow to a different module (configured in the **Exit Module** field) or return a special result.

> **Important**
>
> When a slot is filled successfully or if the intent is switched, the counter is reset to zero.

When the **Enforce Maximum Attempts for Intent Disambiguation** option is configured, the bot will try to identify an intent from the conversation. When the maximum number of unsuccessful attempts is reached, Intelligent Automation either transfers the callflow to a different module (configured in the **Exit Module** field) or return a special result.

Pass Context Settings

Intelligent Automation allows you to pass variables to an NLU engine. The NLU engine can use the variables as part of the slot filling capabilities when the **Send Intelligent Automation variables as context to NLU Engine** setting is enabled.

For DialogFlow, the parameter's default value must be mapped to the variable defined in Intelligent Automation. You can configure the Default Value option from the contextual menu for a parameter in DialogFlow and it should be use the following format: *#fish_context.<VariableName>*.

Bypass Opening Question

You can allow users to bypass the initial question from the bot by passing the utterance as a variable to the NLU. The NLU then skips the initial question and proceeds to the next question. You can use any previously collected data or create and define a variable in the Script block and pass the utterance as a parameter in the **Link** block to the Natural Language Menu. The utterance is then passed to the NLU menu as an input in the callflow.

Support for Follow-up Intents

Intelligent Automation now supports follow-ups intents from Google Dialogflow. A follow-up intent is an intent that is tied to another earlier intent. Think of it as a child intent of an existing parent intent. When you create a follow-up intents in Dialogflow, Intelligent Automation can read and work with contexts for the intents.

Set up a module with a **Question** block. The Question block stores the intent in the **Prompt Wording** field. Ensure that this field has `var:NLTextResponse` as its value. In the **Question Options** tab, enable the **Store Answer as a variable** option. This holds the values received from DialogFlow. Ensure that the variable name is `utterance`.

Support for Fallback Intents

If a fallback intent is detected, IA will refer to the **Ask Natural Language Question** and return the fallback text that was configured in Dialogflow. This text could be something like: *Sorry, I wasn't able to understand that. Can you try explaining again?*. Users can make another attempt to provide information.

The **Enable Maximum Attempts Counters** and **Enforce Maximum Attempts for filling Mandatory Slot** options allow counting the number of attempts and either return a result or redirect to a different module.

Support Phrase Hints

See Phrase Hints for more information.

Support for System entities

> **Important**
>
> This feature is supported only for Google Dialogflow.

DialogFlow supports composite entities and returns them as part of the response. To allow use of these entities in a callflow, Intelligent Automation can use the values as a whole object or split the object into simpler components.

For example, in DialogFlow, we have a slot called *Duration* and the data type is `@sys.duration`, DialogFlow returns the following object as a result:

```
{"amount":10,"unit":"min"}
```

Intelligent Automation will store the object as a slot which can be accessed using a **Script** block.

```
Duration  = {"amount":10,"unit":"min"}
```

Intelligent Automation will also split the object as separate sub-slots which can be referenced in a **Question** block (see section Map Slots to questions). You can access each entity as *<slotName_type> = value*.

```
Duration_amount = 10
Duration_unit = min
```

Sample script

This script will parse the value of the variable, NLSlots and log the slot name, value and confidence.

```
def slots = context.getVariable("NLSlots")

slots.entrySet().each
{
    entry ->
    sSlotName = entry.key
    sSlotValue = entry.value
    sSlotValue.entrySet().each
    {
        valueEntry ->
        value = valueEntry.key
        fConfidence = valueEntry.value
        context.log("Slot info: Name: " + sSlotName + " Value: " + value.getValue() + "
Confidence " + fConfidence)
    }
}
```